Implementació d'un sistema de fitxers dinàmic per containers

Grau en Enginyeria Informàtica

Tecnologies de la Informació

Autor: Guillem Burgués Miró

Director: Alex Pajuelo - Arquitectura de Computadors

Codirector: Xavier Verdú - Arquitectura de Computadors

Data defensa: 24 de Gener del 2017

Facultat d'Informàtica de Barcelona (FIB)

Universitat Politècnica de Catalunya (UPC) - BarcelonaTech

Dades del Projecte

Títol del projecte: Implementació d'un sistema de fitxers dinàmic per containers

Nom de l'estudiant: Guillem Burgués Miró

Director: Alejandro Pajuelo González

Codirector: Xavier Verdú Mulá

Departament director: Arquitectura de Computadors

Data de defensa: 24 de Gener del 2017

Titulació: Grau en Enginyeria Informàtica

Especialitat: Tecnologies de la Informació

Centre d'estudis: Facultat d'Informàtica de Barcelona

Universitat: Universitat Politècnica de Catalunya

Membres del Tribunal

President: Ruben Tous Liesa

Vocal 1: Daniel Jimenez Gonzalez

Vocal 2: Miquel Noguera Batlle

Vocal Suplent: Jordi Guitart Fernandez

Qualificacions

Qualificació numèrica:

Qualificació descriptiva:

Agraïments

Primer de tot, voldria agrair a la meva família i amics tot el suport incondicional que m'han anat donant al llarg de tot el projecte animant-me a continuar endavant.

També vull agrair als meus directors del treball, Alex i Xavier, per tota l'ajuda que m'han ofert al llarg del projecte per poder tirar-lo endavant resolent-me els dubtes que m'anaven sortint, i per haver-me brindat l'oportunitat d'ajudar-los en un projecte.

Finalment, m'agradaria agrair als meus companys de feina que des del primer moment em van oferir la seva ajuda per a qualsevol dubte que tingués.

Resum

Aquest document conté la memòria del treball de fi de grau d'Enginyeria Informàtica de la Universitat Politècnica de Catalunya (Facultat d'Informàtica de Barcelona -FIB).

L'objectiu d'aquest projecte consisteix a implementar un sistema de fitxers dinàmic per un container a partir d'un sistema ext2 estàtic que actualment està integrat dins del container. La solució software implementada permet realitzar les mateixes operacions que el sistema de fitxers estàtic (crear, llegir, escriure, esborrar, tancar) a més a més de créixer dinàmicament eliminant així les restriccions que suposa un sistema de fitxers estàtic com és la limitació de la quantitat d'informació que poden guardar, entre altres coses.

Resumen

Este documento contiene la memoria del trabajo de fin de grado de Ingeniería Informática de la Universidad Politécnica de Catalunya (Facultad de Informática de Barcelona - FIB).

El objetivo de este proyecto consiste en implementar un sistema de ficheros dinámico para un container partiendo de un sistema de ficheros ext2 estático que actualmente está integrado en este container. La solución software implementada permite realizar las mismas operaciones que el sistema de ficheros estático (crear, leer, escribir, borrar, cerrar) con el añadido de que puede crecer dinámicamente eliminando así las restricciones que supone un sistema de ficheros estático como es la limitación de la cantidad de información que pueden guardar, entre otras cosas.

Abstract

This document contains the final degree work's memory of Computer Engineering (Barcelona School of Informatics - Polytechnic University of Catalonia).

The main objective of this project is to implement a dynamic file system for a container starting from a static file system ext2 that it's currently integrated in this container. The implemented software solution offers the same usefulness as the static ext2 does (create, read, write, delete, close) as well as growing dynamically eliminating the restrictions that goes with the static file systems as the limitation of the amount of data that they can store among other things.

Índex

Índex de taules

Índex de figures

1. Introducció i contextualització

Aquest projecte és un Treball Final de Grau de modalitat A de l'especialitat Tecnologies de la Informació. S'ha desenvolupat a la Facultat d'Informàtica de Barcelona (Universitat Politècnica de Catalunya) al Departament d'Arquitectura de Computadors.

1.1 Context

En l'àmbit de la informàtica, un container proveeix un nivell de virtualització del sistema operatiu que permet executar múltiples aplicacions (aïllades entre elles) en un mateix sistema [1].

Els containers inclouen diversos beneficis: permeten la creació d'aplicacions i el seu desplegament d'una manera molt ràpida i eficient, proporciona aïllament de recursos, eficiència en la utilització de recursos, portabilitat tant en sistemes operatius com en el *cloud...* [1].

Amb la definició donada a l'inici, es podria caure en l'error de pensar que les màquines virtuals (MV) i els containers són el mateix, però diferències en l'arquitectura permeten als containers ser més eficients.

Les màquines virtuals virtualitzen tot el hardware, inclouen l'aplicació, els binaris, les llibreries necessàries i, a més a més, un sistema operatiu convidat on s'executa l'aplicació. Tot això implica una mida de GBs considerable.

En canvi els containers inclouen l'aplicació, les dependències (llibreries, binaris...) però comparteixen el kernel amb altres containers, executant-se com a processos aïllats en l'espai d'usuari del sistema operatiu host [2]. En la figura 1 es poden apreciar les diferències en l'arquitectura d'una màquina virtual amb un container Docker. El fet que en l'arquitectura dels containers aparegui Docker Engine però jo em refereixi a ells com a container Docker és perquè el Docker Engine és la part del Docker que s'encarrega de crear i executar els containers Docker.

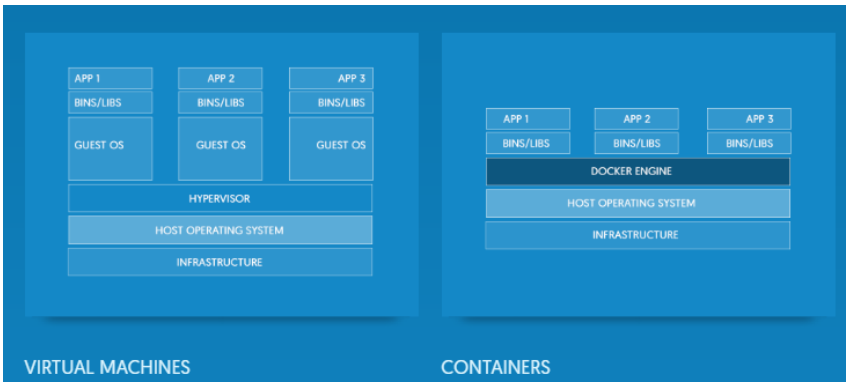


Figura 1. Comparativa de MV amb Containers Docker

Una part important dels containers són els sistemes de fitxers. Un sistema de fitxers conté les estructures de dades i els mètodes que un sistema operatiu utilitza per gestionar l'espai i localitzar els arxius d'un disc o d'una partició; és a dir, és la manera en què els arxius són organitzats en un disc [3].

Qualsevol sistema de fitxers té una estructura interna. Els més coneguts són NTFS i els d'UNIX.

Més endavant s'explicarà com és aquesta estructura interna i quina és la funció de cada part.

1.1.1 Containers

La història dels containers es remunta a l'any 1979 amb l'aparició de la crida a sistema *chroot*, però no és fins a l'any 2000 que apareix la primera implementació anomenada Jails [4]. Podem veure que els containers són una tecnologia antiga.

Avui dia els containers són un software que està a l'alça a causa de totes les millores que està aportant en el món de la informàtica.

Els containers han introduït una nova manera d'executar les aplicacions.

Per desplegar una aplicació, aquesta s'hi ha d'instal·lar al sistema operatiu *host* mitjançant l'administrador de paquets del mateix sistema operatiu. Això comporta un inconvenient i és que tots els executables de les aplicacions, així com la seva configuració, llibreries... es molesten entre elles i el mateix sistema operatiu host.

Els containers solucionen aquest problema perquè estan basats en una virtualització a nivell de sistema operatiu en lloc de virtualització de hardware. Els containers estan

aïllats entre ells i no es veuen, per tant cada container pot tenir el seu propi sistema de fitxers, entre altres coses.

A continuació es parla breument de Linux Containers (LXC) perquè és la base més utilitzada avui en dia a l'hora d'implementar containers.

Containers - Linux Containers (LXC)

Els linux containers ofereixen un entorn el més semblant possible al que obtindries en utilitzar una maquina virtual però sense el overhead que provoca executar un kernel per separat i simular tot el hardware perquè els containers només repliquen aquells components necessaris per funcionar. Permeten executar múltiples sistemes operatius aïllats entre sí. La primera versió que va sortir data del 6 d'agost del 2008 [5].

Linux containers aconsegueix això mitjançant una combinació de característiques de seguretat del kernel com són el namespace i el control groups (cgroups) que aïllen els contenidors entre ells i dels procesos executant-se en el host.

La funció del namespace és agafar un conjunt de recursos del sistema i ensenyar-li al procés del container de manera que aquest els vegi com si estiguessin dedicats exclusivament per ell.

La funció del cgroup s'encarrega de l'aÏllament i la utilització dels recursos del sistema per a un conjunt de procesos. És a dir, si una aplicació agafa una quantitat de cicles de CPU i memòria molt gran, es pot posar en un cgroup per limitar l'ús de CPU i memòria.

En resum, cgroups gestiona els recursos per un conjunt de processos i namespace per un únic procés.

1.1.2 Sistemes de fitxers

En aquest apartat parlarem del sistema de fitxers ext2 perquè en aquest treball s'implementa un ext2 dinàmic i del NTFS perquè és dels més coneguts i així veiem maneres diferents en com s'estructuren.

Sistemes de fitxers - EXT2

Ext2 és un sistema de fitxers UNIX i la seva primera versió va ser introduïda l'any 1993 per el kernel 0.99 de Linux [6].

Els sistemes de fitxers d'UNIX tenen una estructura general molt semblant tot i que els detalls exactes poden variar d'un a l'altre [3]. Per fer una introducció de l'estructura s'ha escollit l'ext2 que és el tractat en aquest treball.

El primer sector conte tota la informació del *boot* del sistema. Com es veu en la figura 3, ext2 divideix el disc en grups de blocs, on a cada grup trobem el *Superblock*, descriptors de grups, bitmap de bloc de dades, taula d'inodes, bitmap d'inodes i els blocs de dades.

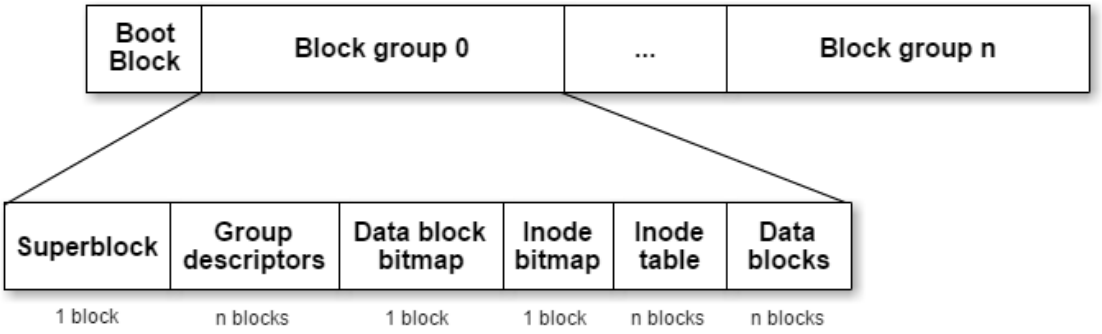


Figura 2. Partició amb ext2

A continuació s'explica breument la funció de cadascun.

Superblock

Conté tota la informació del sistema de fitxers en conjunt com la mida del bloc, nombre de blocs, número de inodes, identificador de sistema de fitxers, número de blocks/inodes lliures... [6]

Descriptors de grups

Conté informació del grup de bloc corresponent. En concret guarda en quin número de bloc està el bitmap de bloc de dades, en quin bloc està el bitmap d'inodes i en quin bloc comença la taula d'inodes. A més a més també guarda el número de bloc i inodes lliures.

Bitmap de bloc de dades

Estructura de dades que conté quins blocs estan utilitzats i quins no.

Taula d'inodes

Són blocs consecutius de memòria que guarden els inodes. Un inode és una estructura de dades dels sistemes de fitxers que emmagatzema les característiques (permisos, mida...) d'un fitxer, directori o qualsevol altre objecte que es trobi dins del sistema de fitxers [6]. Cada fitxer té un inode i aquest està identificat amb un únic número a la taula de inodes. Dins dels inodes trobem una estructura amb punters que apunten a altres punters o als blocs de dades.

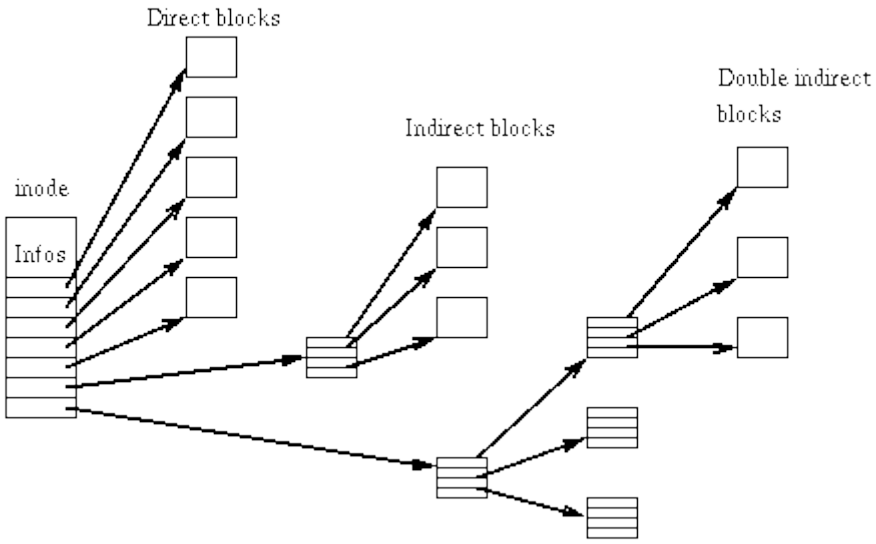


Figura 3. Estructura d'un Inode

Bitmap d'inodes

Estructura de dades que conté quins inodes estan utilitzats i quins no.

Blocs de dades

Blocs de dades on es guarda el contingut dels arxius, directoris...

Sistemes de fitxers - NTFS

NTFS és un sistema de fitxers de WIndows NT i la seva primera versió va ser introduïda l'any 1993 per Windows NT 3.1 [7].

L'estructura del NTFS conté dos sectors de *boot*: un a l'inici i una còpia d'aquest al final de la partició. Aquest sector, com el seu propi nom indica, generalment serveix per arrencar el sistema operatiu.

En NTFS, tots els arxius amb la informació relacionada, directoris... són guardats com metadades en la Master File Table (MFT) [8]. Quan augmentem el volum de dades en un sistema de fitxers NTFS aquest ho va registrant en el MFT i va creixent. Per tal de poder créixer hi ha una zona reservada en l'estructura coneguda com a MFT zone. On acaba la MFT zone trobem ja tots els fitxers que tenim en el nostre sistema.

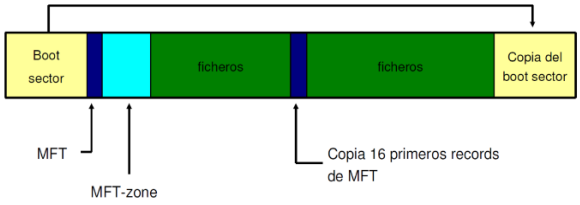


Figura 4. Partició amb NTFS

1.2 Actors implicats

Com en tots els projectes que es realitzen, hi ha una quantitat de persones implicades (directe o indirectament).

- Desenvolupador

  És la persona més important d'aquest projecte perquè s'encarrega de desenvolupar-lo per tal d'assolir l'objectiu proposat. A més d'oferir la solució software, és l'encarregat de redactar, preparar i exposar tota la memòria del projecte.

- Director i subdirector

  El director i subdirector són en Alex Pajuelo i en Xavier Verdú respectivament. Són dues persones molt importants en el projecte perquè realitzen tasques com assessorar al desenvolupador en la presa de decisions al llarg de tot el projecte, així com dur a terme el seguiment i monitoratge del TFG per tal que

no es produeixi cap desviament de l'objectiu final i, en cas que això succeeixi, corregir-ho com més aviat millor.

- Administradors de sistemes o usuaris finals

  Qualsevol administrador de sistemes o usuari final es beneficiarà d'aquest nou software.

1.3 Estat de l'art

En aquest apartat parlarem de com són els sistemes de fitxers en els containers.

La forma més utilitzada consisteix a utilitzar el chroot per tenir el sistema de fitxer desitjat. Per començar, explicarem que és el chroot.

Chroot és una operació en els sistemes operatius UNIX que canvia el directori root pel procés que està executant-se i els seus fills [9]. El programa que s'està executant en aquest entorn modificat no pot accedir a fitxers fora del seu directori d'arbres que té assignat. Aquesta crida a sistema va ser introduïda durant el desenvolupament de la versió 7 d'UNIX l'any 1979 [10].

Explicat això, la idea és que el sistema de fitxers estigui en una carpeta dins del sistema host i que aquest mitjançant chroot canviï el directori root al directori on tenim el sistema de fitxers del container. Amb això aconseguim tenir el sistema de fitxers desitjat en el qual podem realitzar tots els canvis que vulguem sense afectar el sistema host.

LXC en lloc d'utilitzar chroot utilitza la comanda pivot_root que s'encarrega de realitzar el canvi del sistema de fitxers del root. Aquesta comanda rep dos parametres (directori nou i directori vell) [11]. El que fa, és moure el sistema de fitxers del procés que el crida al directori vell i en crea un de nou (directori nou) que passarà a ser el nou sistema de fitxers del root que en el nostre cas és el directori on tenim el sistema de fitxers desitjat.

1.4 Formulació del problema

Actualment, un grup de recerca de la UPC està dissenyant una nova arquitectura de containers que superi les prestacions de les actuals. El sistema de fitxers que utilitzen

per desenvolupar el container és un ext2 estàtic (té una mida fixa) que limita tant el nombre de fitxers com la mida que poden tenir aquests a més a més de la mida total i la diferència respecte a altres propostes de containers és que aquest sistema de fitxers està integrat dins del container.

Davant d'aquesta limitació ens trobem en tres casos:

- Mida fixa: tenim una mida determinada i no podem fer res més que adaptar-nos a ella.
- Mida insuficient: si volem executar aplicacions que siguin més grans que la mida que està establerta, ens trobarem en una situació que haurem de crear un de nou i més gran per tal d'executar l'aplicació desitjada.
- Mida massa gran: en aquest cas estem desaprofitant tot l'espai que l'aplicació no utilitza.

L'objectiu d'aquest treball serà implementar una solució software per aquest sistema de fitxers del container per tal que sigui dinàmic i no limitar-lo com succeeix actualment i, en un futur, poder-lo utilitzar en el container que està sent dissenyat en aquests moments.

1.5 Estructura del document

Aquesta memòria està dividida en la introducció (Capítol 1) i 12 capítols més. En el Capítol 2 s'expliquen tots els objectius del projecte, els seus possibles obstacles, així com quin pla de contingència i metodologia se seguirà. En el Capítol 3 es mostra tota la planificació del projecte i els recursos associats a aquest. En el Capítol 4 es parla sobre la gestió econòmica que comporta el projecte. En el Capítol 5 s'explica al lector el sistema de fitxers actual. En el Capítol 6 comença l'especificació del software a desenvolupar. En el Capítol 7 s'entra en detall en la implementació del sistema de fitxers. En el Capítol 8 s'explica la verificació del sistema de fitxers així com el compliment dels seus requisits. En el Capítol 9 es mostra detalladament les desviacions tant en temps com econòmiques del projecte. En el Capítol 10 s'explica la sostenibilitat d'aquest projecte i en el Capítol 11 s'explica si el projecte es veu afectat per les lleis i regulacions. Per acabar, en el Capítol 13 trobem les conclusions del projecte.

2. Abast

2.1 Objectius

Com s'ha explicat prèviament, l'objectiu d'aquest treball consisteix a implementar un sistema de fitxers dinàmic, a partir del sistema ext2 estàtic integrat actualment, pel container. Per tal d'aconseguir-ho, s'hi haurà d'extendre l'estructura interna del sistema de fitxers ext2 per tal que contingui totes les dades necessàries per a ser dinàmic, així com modificar tot el funcionament del sistema per a poder utilitzar aquesta nova estructura. La finalitat de tot això és poder fer créixer el sistema de fitxers sense cap limitació. Per aconseguir aquest objectiu, aquest es pot desglossar en objectius secundaris i que són els que s'han anat seguint a l'hora de desenvolupar el sistema de fitxers:

- Incrementar la mida del fitxer.
- Implementar els nous elements de l'estructura interna.
- Modificar la gestió dels bitmaps del nou sistema de fitxers.
- Implementar les funcions del sistema de fitxers que interactuen amb la nova estructura interna.
- Implementació de la integració en l'estructura general.
- Avaluació del sistema.

Per últim, el sistema de fitxers dinàmic ha de complir un seguit de requeriments:

- El nou sistema de fitxers ha de créixer dinàmicament.
- Ha de ser robust davant possibles inconvenients.
- Ha de ser eficient en espai.

2.2 Possibles obstacles i plans de contingència

En tot projecte sempre existeix algun obstacle que més tard o d'hora apareix i aquest, no és una excepció. Tot seguit s'esmenten els possibles obstacles que es poden trobar.

Restricció temporal

Aquest projecte té una data d'entrega final. Per això qualsevol problema que es trobi durant el desenvolupament s'haurà de gestionar el més ràpid possible per complir els terminis del projecte.

Dificultats a l'hora d'entendre el codi

L'avanç en el desenvolupament del projecte està lligat amb el grau d'aprenentatge del codi amb el qual es parteix a l'inici d'aquest treball. En cas de dificultats, els tutors del projecte em poden aclarir dubtes que puguin anar sorgint al llarg del desenvolupament d'aquest.

2.3 Metodologia i rigor

2.3.1 Mètodes de treball

Per dur a terme aquest projecte, s'ha decidit utilitzar una metodologia dividida en 4 fases: fase inicial, fase d'anàlisi, fase de desenvolupament i fase final.

A cada fase hi trobem la feina a realitzar. S'utilitza una metodologia en cascada per a les fases inicial, anàlisi i final, mentre que la fase de desenvolupament seguirà una metodologia de desenvolupament iterativa i incremental.

- Fase Inicial: GEP
- Fase d'anàlisi: Anàlisi, especificació de requisits i memòria.
- Fase de desenvolupament: Especificació - Implementació - Proves - Memòria
- Fase final: Prova final, memòria final i preparació de la defensa oral.

Un cop finalitzada cada fase s'anirà redactant la memòria del treball.

2.3.2 Eines de seguiment

GitLab [12] és un repositori Git basat en web que permet pujar projectes software i portar un control de totes les versions realitzades fins al moment. És molt semblant a GitHub amb la diferència que GitLab té un major control sobre els repositoris. Aquesta eina permetrà als directors poder anar veient la feina sempre que vulguin i les anotacions del desenvolupador de possibles problemes trobats així com la seva

solució. Per tant els directors sempre tenen accés a tot el projecte en qualsevol moment.

Per garantir el bon camí del projecte i solucionar possibles desviacions com més aviat millor, així com assessorar el desenvolupador, es faran reunions periòdiques amb els directors en funció del progrés del projecte. A banda d'aquestes reunions, al llarg de tot el projecte, es mantindrà contacte en tot moment amb els directors mitjançant correu electrònic per a qualsevol dubte.

A més a més s'utilitzarà GanttProject [13] per garantir que s'estan complint tots els terminis que es van decidir a l'inici del projecte.

2.3.3 Mètode de validació

Com s'ha explicat anteriorment, les reunions amb els directors serviran per validar que la feina feta fins al moment és l'adequada.

Un cop realitzada tota la implementació de la solució software, es procedirà a sotmetre-la a una sèrie de proves exhaustives per tal de garantir el seu bon funcionament i detectar possibles errors en aquesta.

Per reduir la quantitat d'errors que es puguin donar a la versió final es faran proves després de la implementació de cada funcionalitat.

3. Planificació temporal

El projecte té una durada definida d'aproximadament cinc mesos: començant al Setembre del 2016 amb Gestió de Projectes (GEP) i acabant a finals de Gener del 2017, el dia de la defensa.

Cal tenir en compte que aquesta és una planificació temporal i que el temps de dedicació per a cada tasca pot variar segons les dificultats que es trobin pel camí.

3.1 Descripció de les tasques

El projecte està definit en 5 fases i aquestes estan subdivididies en tasques. A continuació s'especifica la feina a realitzar en cadascuna de les tasques de cada fase.

3.1.1 Fase prèvia

Consisteix a preparar l'entorn de treball per a dur a terme el projecte i aprendre com funcionen les eines que s'utilitzaran a l'hora de desenvolupar el projecte. No té cap dependència.

3.1.2 Fase inicial

Documentació GEP

Aquesta tasca es realitzarà a principis de Setembre i durarà aproximadament un mes i mig. Durant aquest temps s'anirà fent i entregant tota la documentació del projecte necessària per assolir els 3 crèdits ECTS corresponents a GEP.

Presentació GEP

Preparació de la presentació final de GEP que es realitzarà un cop finalitzada la documentació.

Un cop acabades aquestes dues tasques, una part de la memòria ja estarà feta a falta dels canvis que es puguin realitzar a partir dels comentaris dels avaluadors. Aquesta fase no té cap dependència.

3.1.3 Fase d'anàlisi

En aquesta fase s'analitzarà tot el codi per adquirir coneixements de com funciona. Es duran a terme reunions amb els directors per tal de resoldre possibles dubtes que es puguin trobar durant l'anàlisi i així garantir el bon camí del projecte. Un cop realitzat aquesta anàlisi amb els dubtes resolts, s'especificaran els requisits i s'escriurà a la memòria. Aquesta fase no té cap dependència.

3.1.4 Fase de desenvolupament

Té com a dependència la fase d'anàlisi perquè sense ella no podríem començar a desenvolupar, ja que no sabríem els requisits què necessita el sistema ni com funciona aquest.

L'objectiu d'aquest projecte és implementar un sistema de fitxers dinàmics per containers i per dur a terme aquesta feina s'ha decidit dividir la fase de desenvolupament en dues tasques i aquestes estan dividides en quatre subtasques.

Implementació del sistema de fitxers dinàmic

És una de les tasques més importants, on s'implementa part de l'objectiu d'aquest projecte.

Primer es realitzarà l'especificació d'aquesta part, per a continuació implementar el codi necessari perquè el sistema de fitxers sigui dinàmic. Un cop realitzades aquestes dues subtasques es procedirà a realitzar una tercera per provar que la implementació compleix amb els requeriments del projecte i, finalment, s'escriurà tota la informació corresponent a la memòria final.

Implementació de l'estructura interna del sistema de fitxers

Aquesta tasca es podria realitzar abans de l'anterior o inclús en paral·lel però es va acordar amb els directors del projecte realitzar-la després perquè és més fàcil integrar el codi un cop feta la part dinàmica.

Igual que en l'anterior tasca, es procedeix a fer una especificació i després implementar el nou codi. Un cop acabades aquestes subtasques es tornarà a realitzar unes proves per garantir que la feina feta fins aquest punt segueix funcionant com s'espera i per comprovar si la nova implementació compleix amb la seva finalitat.

Un cop es doni per acceptada la solució, s'escriurà tota la informació d'aquesta a la memòria final.

3.1.5 Fase final

En la fase final es realitzaran un seguit de tasques.

Prova final

Consisteix a passar-li al projecte un seguit de proves exhaustives per tal de verificar que la fase d'implementació s'ha realitzat bé i compleix amb els requisits especificats anteriorment.

Memòria

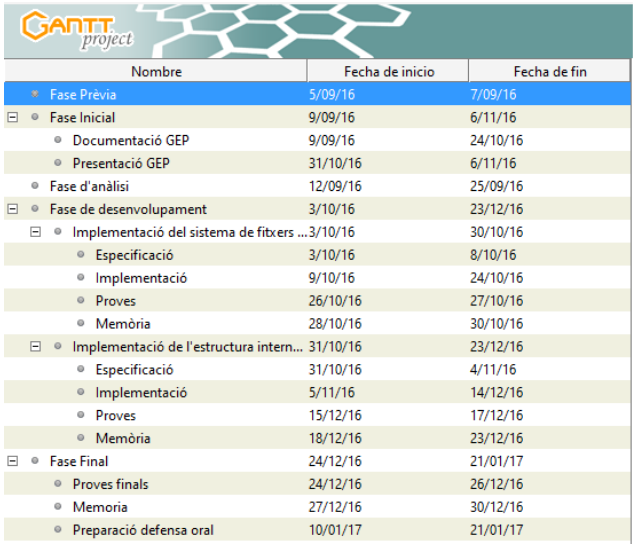
Un cop realitzada la prova final es procedirà a acabar la memòria.

Preparació defensa oral

Per últim però no menys important, es prepararà la defensa oral del projecte perquè sigui realitzada de la millor manera possible.

3.2 Diagrama de Gantt

A continuació es mostra en detall tota la planificació del projecte. En la figura 5 es pot observar totes les fases i tasques que s'han comentat en l'apartat anterior detallades amb la data d'inici i finalització, així com l'ordre en què es realitzen.



Nombre	Fecha de inicio	Fecha de fin
Fase Prèvia	5/09/16	7/09/16
Fase Inicial	9/09/16	6/11/16
Documentació GEP	9/09/16	24/10/16
Presentació GEP	31/10/16	6/11/16
Fase d'anàlisi	12/09/16	25/09/16
Fase de desenvolupament	3/10/16	23/12/16
Implementació del sistema de fitxers ...	3/10/16	30/10/16
Especificació	3/10/16	8/10/16
Implementació	9/10/16	24/10/16
Proves	26/10/16	27/10/16
Memòria	28/10/16	30/10/16
Implementació de l'estructura intern...	31/10/16	23/12/16
Especificació	31/10/16	4/11/16
Implementació	5/11/16	14/12/16
Proves	15/12/16	17/12/16
Memòria	18/12/16	23/12/16
Fase Final	24/12/16	21/01/17
Proves finals	24/12/16	26/12/16
Memoria	27/12/16	30/12/16
Preparació defensa oral	10/01/17	21/01/17

Figura 5. Planificació del projecte

En la figura 6 es mostra el diagrama de Gantt del projecte on es poden observar quines tasques es van realitzan en paral·lel (sempre que es pugui) i les dependències entre elles.

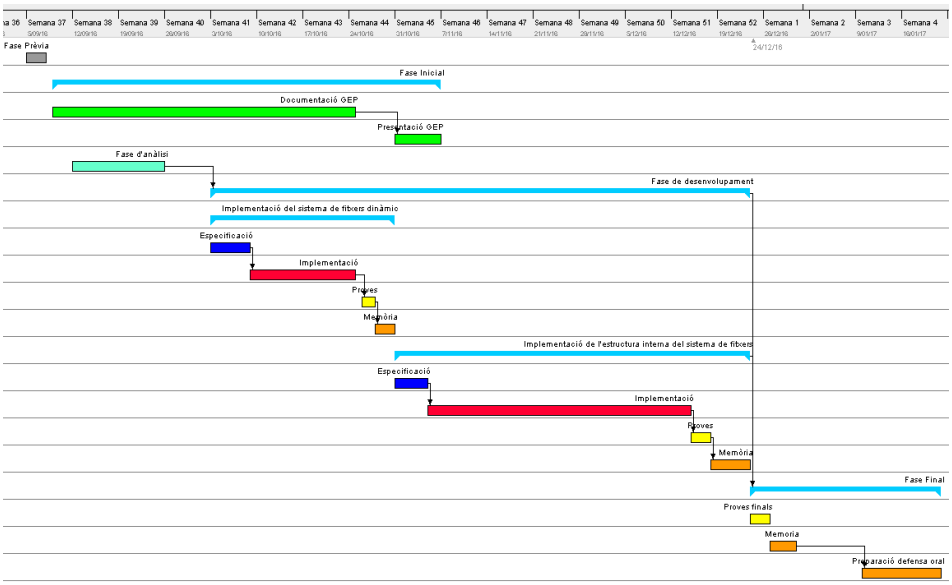


Figura 6. Diagrama de Gantt

3.3 Recursos

En tot projecte sempre es necessiten mitjans per poder portar-lo a bon port. Aquests són recursos humans, de software i de hardware. Per a la realització d'aquest projecte s'utilitzaran els següents:

3.3.1 Recursos humans

- Estudiant d'enginyeria informàtica especialitzat en Tecnologies de la Informació que realitza la tasca de desenvolupar aquest projecte.
- Director i co-director que s'encarreguen de fer el seguiment del projecte per assegurar la bona finalització d'aquest i assessorar a l'estudiant.

3.3.2 Recursos hardware

- Ordinador de sobretaula amb Windows 10 de 8 GB de memòria RAM amb connexió a Internet per desenvolupar el projecte.
- Ordinador portàtil amb Windows 7 de 4 GB de memòria RAM amb connexió a Internet per desenvolupar el projecte.

  El segon ordinador és de reserva perquè, en cas que el primer tingui una avaria, no repercuteixi en la planificació del projecte.

3.3.3 Recursos software

- Microsoft Office 2007 per a realitzar GEP i la memòria final del treball.
- Microsoft Visual Studio Community 2015 com a entorn de programació
- GitLab pel control de versions.

3.4 Valoració d'alternatives i pla d'acció

Com s'ha comentat a l'inici de la planificació, aquesta es pot veure afectada per desviacions en algun moment donat al realitzar certes tasques.

Ens podem trobar en una situació que les tasques de la fase de desenvolupament es realitzen més ràpidament (cosa que no comporta cap problema) o bé que necessitin més hores de les planificades. En cas que una tasca comporti més temps del necessari es començarà la següent més tard. Si el retràs és significatiu, el desenvolupador li dedicarà més hores per finalitzar la tasca com més aviat millor i així complir amb el termini final, amb la qual cosa incrementaria el cost de recursos humans. Per tal d'evitar aquest retràs, s'ha establert un marge considerable de temps entre la finalització de la fase de desenvolupament i la data límit d'entrega per tal d'assegurar la finalització del projecte.

4. Gestió econòmica

Després de planejar tot el projecte i dissenyar la solució cal realitzar un estudi econòmic i d'impacte ambiental i social per determinar si és un projecte viable o no. Per dur a terme això, el primer que cal fer és calcular els costos dels recursos necessaris per al desenvolupament del projecte.

4.1 Consideracions inicials

Abans de començar amb l'estudi econòmic cal esmentar que es parteix de la base de què no es disposa del software de Microsoft gratuït proporcionat per la UPC com a estudiant d'aquesta i tindrem en compte els preus reals.

Per obtenir els costos, s'ha tingut en compte que es treballaran 4 hores diàries aproximadament per cada dia de dedicació.

4.2 Identificació i estimació dels costos

Per fer una estimació dels costos dels recursos, primer cal identificar-los. En aquest projecte s'utilitzaran recursos humans, de hardware, de software i altres. En l'estimació, es té en compte el pla de contingència i cost de possibles imprevistos.

4.2.1 Recursos humans

Per calcular el cost en recursos humans, s'ha consultat l'informe de Page Personnel [14] de l'actual any (2016-2017). En la següent taula es mostra per cada rol del projecte, les hores destinades que dedicarà cadascun, el seu salari per hora i total, així com el cost final dels recursos humans.

Rol	Hores	€/h	Total
Cap de projecte	287	20	5.740 €
Analista	56	18,2	1.019.2 €
Dissenyador	44	17,18	755,92 €
Programador	224	16,66	3.731,84 €
Tester	32	13,54	433,28 €

Total	643		11.680,24 €

Taula 1. Cost total recursos humans

4.2.2 Costos directes per activitat

Tot seguit es mostra els costos que comporten totes les activitats del projecte desglossades:

Activitat	Recursos	Hores	Cost
Fase Inicial			
Documentació GEP	Cap de projecte	188	3.760 €
Presentació GEP	Cap de projecte	28	560 €
Fase d'anàlisi	Analista	56	1.019,2 €
Fase de Desenvolupament **Implementació del sistema de fitxers dinàmic**			
Especificació	Dissenyador	24	412,32 €
Implementació	Programador	64	1.066,24 €
Proves	Tester	8	108,32 €
Memòria	Cap de projecte	12	240 €
Fase de Desenvolupament **Implementació de l'estructura interna**			
Especificació	Dissenyador	20	343,6 €
Implementació	Programador	160	2.665,6 €
Proves	Tester	12	162,48 €
Memòria	Cap de projecte	24	480 €
Fase final			
Proves finals	Tester	12	162,48 €
Memòria	Cap de projecte	24	480 €
Preparació defensa oral	Cap de projecte	11[1]	220 €
Total		643	11.680,24 €

Taula 2. Costos directes per activitat

[1] Durant la preparació de la defensa oral només es dedicarà una hora al dia.

4.2.3 Costos indirectes

Costos derivats de la utilització de materials i/o dispositius per realitzar aquest projecte.

Software

A continuació es mostra una taula amb el software que s'utilitzarà en aquest projecte així com les unitats i els preus.

Producte	Preu	Unitats	Cost
Microsoft Visual Studio 2010	0 €	1	0 €
Windows 10 Pro	279 €	1	279 €
Windows 7	146 €	1	146 €
Microsoft Office 2007	0 €	1	0 €
Total			425 €

Taula 3. Costos Software

Hardware

La següent taula mostra el hardware que es farà servir durant el projecte, el seu cost, les unitats i l'amortització (50%).

Producte	Unitats	Vida útil	Cost	Amortització
Ordinador de sobretaula	1	5 anys	850 €	85 €
Ordinador Portàtil	1	3 anys	325 €	54.16 €
Total			1.175 €	139,16 €

Taula 4. Costos Hardware

Transport

S'utilitzarà el tren (RENFE) amb un abonament trimestral de 3 zones amb un cost de **199,20 €**

Impressions en paper

En finalitzar el projecte, cal entregar en paper tota la documentació d'aquest a cadascun dels membres del tribunal. Per calcular el seu cost, suposarem que tindrà una extensió d'aproximadament 100 pàgines per memòria, tres membres del tribunal i amb un cost de 0,05 € per pàgina amb enquadernació inclosa. Per tant el cost de les impressions serà de 15 €.

Costos indirectes totals

La següent taula mostra un resum dels costos indirectes així com el cost total d'aquests.

Recurs	Cost
Software	425 €
Hardware	139,16 €
Transport	199,20 €
Impressions en paper	15 €
Total	778,36 €

Taula 5. Costos totals indirectes

4.2.4 Contingència

S'ha decidit reservar una part del pressupost per al pla de contingència. En concret un 15% de la suma de costos directes i indirectes.

Costos	Percentatge	Preu	Cost
Directes	15 %	11.680,24 €	1.752,036 €
Indirectes	15 %	778,36 €	116,754 €
Total			1.868,79 €

Taula 6. Costos de contingència

4.2.5 Imprevistos

A l'hora d'elaborar el pressupost del projecte, s'ha tingut en compte dos possibles imprevistos:

- **Avaria del hardware**

  En qualsevol moment del projecte es podria donar la situació que el hardware tingués algun problema. En tenir dos ordinadors disponibles per treballar garanteix que si falla un l'altre està a lliure disposició. En cas que els dos tinguin una avaria, caldria reparar un d'ells o comprar-ne un de nou si fos més econòmica aquesta última opció. Aquest esdeveniment li assignem una probabilitat del 3%.

- **Retard d'una setmana**

  Com es menciona en l'apartat de planificació, es deixa un marge de temps per tal de gestionar possibles desviacions en la fase de desenvolupament. Aquest marge concretament consta d'una setmana durant la qual el programador haurà de treballar amb un cost de 16,66 €/hora. Li assignem una probabilitat del 10%.

Imprevist	Probabilitat	Unitats	Preu	Cost
Retard d'una setmana	10 %	28 hores	16.66 €/hora	466,48 €
Avaria Hardware	3 %	1	375 €	375 €
Total				841,48 €

Taula 7. Costos Imprevistos

4.2.6 Pressupost total

Concepte	Cost
Costos directes	11.680,24 €
Costos Indirectes	778,36 €
Contingència	1.868,79 €

Imprevistos	841,48 €
Total	15.168,87 €

Taula 8. Pressupost Total

4.3 Control de gestió

La part de recursos humans és la que més pot variar al llarg de tot el projecte i, per tant, caldrà fer un control constant per evitar un increment considerable en el pressupost. Aquest control consistirà a anar comparant la data de finalització prevista de les tasques previstes amb la data de finalització real. Es mirarà el cost calculat inicialment segons les hores planificades amb el cost de les hores que s'ha treballat realment. La fase que més pot variar el pressupost inicial amb les seves possibles desviacions és la de desenvolupament. Per realitzar aquest control, es disposa d'una taula Excel en la qual cada dia s'apuntaran les hores dedicades per cada rol.

5. Sistema de fitxers actual

Abans d'entrar de ple en les especificacions del sistema de fitxers dinàmic és necessari explicar com és el sistema de fitxers actual per poder entendre els canvis que s'han realitzat amb l'objectiu de fer-lo dinàmic.

La següent figura presenta l'arquitectura del sistema de fitxers actual així com la comunicació que es realitza entre les parts.

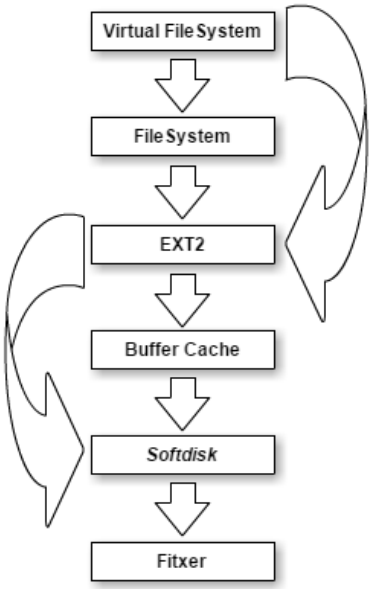


Figura 7. Estructura general

Virtual File System - File System és la capa d'abstracció que es situa a sobre dels sistemes de fitxers. Això ens dóna la possibilitat que les aplicacions accedeixin a diferents sistemes de fitxers d'una manera uniforme. En el nostre cas ens proporcionarà accés a l'ext2 estàtic i al dinàmic.

Aquesta classe té mètodes que es comuniquen amb l'ext2 (funcions de lectura, escriptura, obrir fitxer...). A més a més aquí es troben la taula de canals, la taula de fitxers oberts i la taula d'inodes.

La taula de canals conté la relació fitxer - descriptor de fitxer, la taula de fitxers oberts té tantes entrades com fitxers oberts amb un índex per a cadascun d'ells i per últim tenim la taula d'inodes, que conté la informació dels inodes que estan actius.

El nombre d'entrades per aquestes taules el defineix l'administrador en el moment de generar el sistema.

Ext2 és tota la implementació del sistema de fitxers. Aquí es troben totes les operacions que es poden aplicar a un fitxer així com les necessàries per poder-les dur a terme. La classe es comunica amb el *Softdisk* per poder realitzar escriptures o lectures a disc directament.

Entre l'ext2 i el *Softdisk* trobem la buffer cache que s'encarrega de guardar els blocs de dades que han estat sol·licitats prèviament per tal d'agilitzar les lectures i les escriptures. La Buffer Cache es comunica amb el *Softdisk* quan les dades que conté s'han d'escriure a disc.

Per últim, la funció del *Softdisk* és crear el fitxer en el qual anirem emmagatzemant el sistema de fitxers, l'escriptura i lectura a disc dels blocs i el mount i unmount del sistema de fitxers.

Un cop explicat l'estructura general on funciona el sistema de fitxers, cal donar detalls de com és l'estructura interna del ext2 per poder entendre els canvis necessaris per implementar el sistema de fitxers dinàmic.

L'estructura del sistema de fitxers estàtic del qual es parteix, està formada pels següents elements:

- *Master Boot Record*: conté la informació necessària per iniciar el sistema.
- *Superblock*: Informació del sistema de fitxers.
- Bitmap Blocs: indica quins blocs del sistema de fitxers estan sent utilitzats.
- Bitmap Inodes: indica quins inodes del sistema de fitxers estan sent utilitzats.
- Blocs Inodes: blocs destinats a inodes.
- Blocs Dades: blocs destinats a dades.

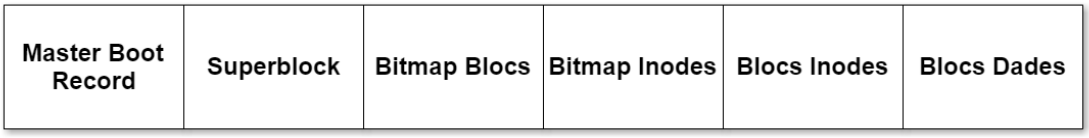


Figura 8. Estructura ext2 estàtic

Aquest sistema de fitxers permet la modificació de certs paràmetres com la mida del bloc, la mida dels sectors de disc així com el nombre de sectors d'aquest.

Tots aquests paràmetres determinen la quantitat de blocs dels quals disposarà el sistema de fitxers fins a un màxim de 8GB permesos. Els paràmetres configurables pel sistema de fitxers són els següents:

Mida de bloc: 512 Bytes, 1KB, 2KB, 4KB, 8KB.

Mida de sector de disc: 512 Bytes, 1KB, 2KB, 4KB, 8KB.

Nombre de sectors: fins a un màxim de 8 GB.

Com es pot veure en la figura 8, els blocs d'inodes estan localitzats entre el Bitmap i els blocs de dades, amb la qual cosa un cop utilitzats tots ja no podem tenir més. El mateix cas són els blocs de dades, limitats pels blocs d'inodes i pel final del mateix fitxer.

El no poder incrementar-los és el que fa que el nostre sistema de fitxers sigui estàtic.

A continuació es mostra un exemple per acabar-ho d'entendre.

Tenim un sistema de fitxers ext2 amb els següents paràmetres:

- Mida Bloc: 4KB
- Mida Sector disc: 512 B
- Nombre de sectors: 512

Per saber de quants blocs de dades disposem en el nostre sistema de fitxers cal determinar primer el nombre de sectors per bloc:

Sectors per bloc = Mida Bloc / Mida Sector disc = 4096 / 512 = 8 sectors.

Un cop sabem que a cada bloc li corresponen 8 sectors:

Nombre de blocs = Nombre de sectors / Sectors per bloc = 64 blocs totals en el sistema de fitxers.

D'aquests 64 blocs, 2 són pel *Master Boot Record* (MBR) i el *Superblock* amb la qual cosa queden 62 blocs pels inodes, les dades i els bitmaps per gestionar aquests últims.

La mida de blocs ocupada pel bitmap de blocs es calcula a partir del nombre de bytes que ocupa:

Bitmap en bytes = (Nombre de blocs / 8 bits per byte) + 1 = (64 blocs /8 bits*bytes) + 1 = 9 bytes.

És necessari sumar 1 perquè, si tenim un nombre de blocs inferior a 8, obtindríem que el nostre bitmap ocupa 0 bytes quan en veritat no és així. A l'hora de calcular els blocs també és necessari fer-ho per la mateixa raó.

Un cop tenim el nombre de bytes, el nombre de blocs que ocupa són:

Bitmap en blocs = (Bytes/Mida Bloc) + 1 = (9 bytes / 4096 bytes) + 1 = 1 bloc.

En aquest sistema de fitxers estàtic els blocs d'inodes representen un 10% dels blocs totals:

Nombre d'inodes = Nombre de blocs * 0.1 = 64 * 0.1 = 6 blocs per inodes.

Aquí ja tenim la primera limitació en el sistema de fitxers: en tenir 6 inodes només podrem crear 6 fitxers/directoris.

Seguint els mateixos càlculs que pel bitmap de blocs, el d'inodes ocupa:

Bitmap en bytes = (Nombre d'inodes / 8 bits per byte) + 1 = (6 inodes /8 bits*bytes) + 1 = 1 byte, per tant el nombre de blocs que ocupa són:

Bitmap en blocs = (Bytes/Mida Bloc) + 1 = (1 byte / 4096 bytes) + 1 = 1 bloc.

Un cop realitzats tots els càlculs tenim:

Paràmetres	Ocupa
Mida bloc	4096 bytes
Mida sector	512 bytes
Sectors	512
Nombre de blocs	64 blocs
Nombre d'inodes	6 inodes (6 blocs)
Bytes bitmap blocs	9 bytes
Blocs bitmap blocs	1 bloc
Bytes bitmap inodes	1 byte
Blocs bitmap inodes	1 bloc

Taula 9. Exemple dades sistema fitxers

Dels 64 blocs, 1 és pel MBR, 1 pel *Superblock*, 2 pels bitmaps, 6 pels inodes per tant tenim 54 blocs per a les dades.

Com hem dit abans, en aquest cas només podríem tenir 6 fitxers/directoris (realment 5 perquè un és necessari per al root) i això ens limita la quantitat d'informació que podem tenir en aquest sistema de fitxers així com el desaprofitament de blocs de dades que mai s'arribaran a utilitzar a causa de la restricció en el nombre de fitxers (si aquests no són escrits) o bé la inutilització de la resta d'inodes si un fitxer ocupés els 54 blocs que corresponen a dades.

Un cop vista la limitació actual del sistema de fitxers podem passar a explicar l'especificació del dinàmic.

6. Incorporació del sistema de fitxers dinàmic

Abans de la implementació cal realitzar l'especificació corresponent, la qual servirà com a punt de partida. Aquesta especificació conté la informació necessària de les necessitats del projecte i de les seves característiques.

A continuació s'expliquen en detall quines són per aquest sistema de fitxers desenvolupat.

Una de les coses que ha de complir el sistema és la possibilitat d'escollir el sistema de fitxers desitjat. Com s'ha comentat, el Virtual File System / File System és la capa d'abstracció que proporciona a les aplicacions un accés a diferents sistemes de fitxers de manera uniforme.

Per tant, aquesta classe haurà d'oferir-nos aquesta funció. En la figura 9 es pot veure la integració del sistema de fitxers dinàmic en l'estructura general del projecte.

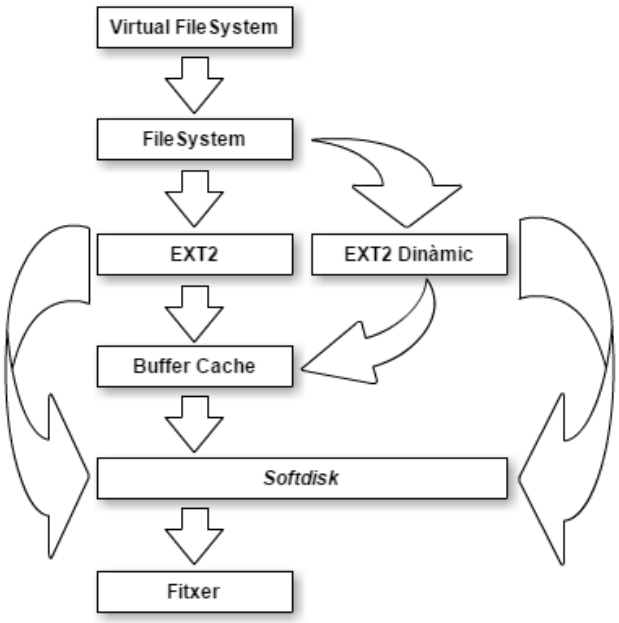


Figura 9. Nova estructura general

6.1 Especificació del sistema de fitxers dinàmic

Aquesta part és la que proporcionarà al sistema de fitxers la possibilitat de ser dinàmic.

Quan el sistema de fitxers ext2 dinàmic detecti que no té més blocs i en necessiti per poder continuar amb les operacions que estigui realitzant, aquest li comunicarà al *Softdisk* que necessita més blocs. Per tant, l'objectiu és que el *Softdisk* demani més memòria al sistema per tal de disposar de més blocs.

Això implica la modificació del nombre de sectors del disc, el nombre de blocs, afegir la nova informació al bitmap de blocs, entre altres coses.

6.2 Especificació de l'estructura interna del sistema de fitxers

Aquesta és la part més important del projecte: la modificació de l'estructura.

Si volem que el sistema de fitxers pugui ser dinàmic no podem seguir utilitzant l'estructura que teníem fins ara per molt que fem crèixer el *Softdisk* perquè l'estructura, i la forma de tractar els bitmaps i els inodes, no serveix perquè està limitada tal com s'ha explicat abans.

Per tal que els bitmaps d'inodes, de blocs, el nombre d'inodes i el nombre de blocs puguin créixer segons les nostres necessitats, és obligatori canviar l'estructura interna del sistema de fitxers. Per aconseguir aquest creixement caldrà tractar els bitmaps i els inodes com a fitxers del sistema i com a tals necessitaran els seus inodes per poder guardar la seva informació. En tractar-los com a fitxers els podrem fer créixer tant com vulguem i d'aquesta manera ja no estem limitats pels paràmetres comentats anteriorment.

Si comparem amb l'estructura de l'estàtic, el nostre sistema de fitxers passarà a tenir els següents elements:

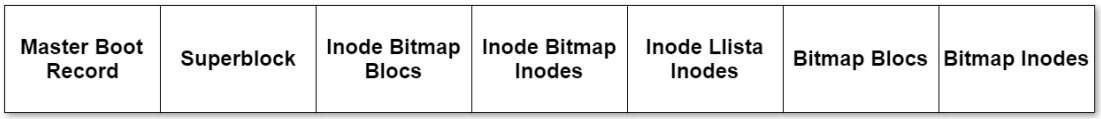


Figura 10. Estructura interna del sistema de fitxers dinàmic

Es pot apreciar l'aparició dels tres inodes especials que gestionen els bitmaps i els inodes i, a més a més, que el sistema de fitxers ocuparà el mínim d'espai possible en el moment de crear-lo. En el sistema de fitxers estàtic aquests inodes especials no eren necessaris perquè no tractàvem els bitmaps com a fitxers perquè no els fèiem créixer però ara sí, per tant els necessitem per a la seva gestió.

A continuació es realitza l'especificació de les accions del sistema de fitxers que es veuen afectades pel canvi en l'estructura interna. Els detalls de la implementació s'expliquen més endavant en el seu corresponent apartat (no es comenten les accions comunes amb l'ext2).

- Formateig del sistema de fitxers: ha de ser capaç de crear els inodes especials i inicialitzar-los als seus valors corresponents. Caldrà que es comuniqui amb el *Softdisk* perquè escrigui la informació d'aquest en el fitxer.

- Muntar el sistema de fitxers: ara a l'hora de muntar-lo caldrà que sigui capaç de llegir els tres inodes especials comunicant-se amb el *Softdisk*.

- Desmuntar el sistema de fitxers: cal que es comuniqui amb el *Softdisk* perquè escrigui la informació dels tres inodes especials per tal que, quan el tornem a muntar, la seva informació sigui correcte.

- Llegir els inodes: el sistema de fitxers ha de ser capaç de llegir (d'un fitxer) els inodes dels directoris i fitxers amb la nova estructura sense cap problema i caldrà que diferenciï entre els inodes especials i la resta.

- Escriure els inodes: ha d'escriure la informació dels inodes (a un fitxer) amb la nova estructura sense cap tipus de problema diferenciant, com quan llegeix inodes, entre els especials i la resta.

- Demanar més blocs: el sistema de fitxers ha de ser capaç de detectar quan no té més blocs disponibles i que només els demani si és realment necessari. Això ha de realitzar-ho quan busqui un bloc lliure. Cal que es comuniqui amb el *Softdisk* per demanar més blocs i un cop aquest li comuniqui que ja en té haurà de gestionar el bitmap de blocs així com actualitzar les variables pertinents a la informació del sistema de fitxers que guarda el *Superblock*. A més a més,

haurà d'escriure en el bitmap de bloc la nova disponibilitat d'aquests i fer-lo créixer.

- Demanar més inodes: caldrà que el sistema de fitxers sigui capaç de detectar que no disposa de més inodes per utilitzar. Quan això succeeixi, ha d'escriure la disponibilitat de més inodes en el fitxer de bitmap corresponent i actualitzar les variables pertinents a la informació del sistema de fitxers que guarda el *Superblock*. No cal que es comuniqui amb el *Softdisk* perquè quan és necessari s'encarrega la funció de demanar més blocs.

Prèviament s'ha fet menció que el *Superblock*, ocuparà el mínim espai possible en el moment de crear el sistema de fitxers amb la qual cosa en el moment de crear-lo, no es realitzarà amb els mateixos valors que l'ext2 dinàmic. Més endavant s'explica amb tot detall.

7. Implementació

En aquesta secció, s'explica en detall la implementació de les funcions del sistema de fitxers especificades en l'apartat anterior.

El sistema de fitxers dinàmic està format per 4 classes: Inode, *Superblock*, Bitmap, Directori i l'ext2 dinàmic.

Les classes inode, bitmap i directori són comunes amb l'ext2 perquè no es veuen afectades pels canvis.

Per altra banda la implementació de l'ext2 dinàmic té les funcions comentades en l'apartat anterior, completament diferents de l'estàtic (la resta no es veuen afectades).

Per últim cal canviar la creació del *Superblock* respecte a com es realitza en l'estàtic.

A continuació s'expliquen els canvis detalladament.

7.1 Creació del *Superblock*

El *Superblock* conté tota la informació del sistema de fitxers. En aquest cas la informació que guardem i utilitzem és la següent:

- *Identifier*: ens permet identificar el tipus de sistema de fitxers (ext2 o ext2 dinàmic)
- *Magic*: el magic number.
- *Major i minor*: Major i minor del sistema de fitxers.
- *SuperBlockSize*: Mida en bytes del *Superblock*.
- *SectorSize*: mida del sector del *Softdisk*.
- *NumberOfBlocks*: Nombre de blocs del sistema de fitxers.
- *NumberOfSectors*: Nombre de sectors en el *Softdisk*.
- *BlockSize*: mida del bloc.
- *NumberOfInodes*: Nombre d'inodes del sistema de fitxers.
- *InodeSize*: Mida en bytes de l'inode (normalment igual a la mida de bloc).
- *FirstInodeBlock & FirstDataBlock*: indica el primer bloc a partir del qual els tenim lliures.
- *InodesBitmapSize*: mida en bytes del bitmap d'inodes.
- *IndoesBMBlocks*: mida en blocs del bitmap d'inodes.

- *BlocksBitmapSize*: mida en bytes del bitmap de blocs.
- *BlocksBMBlocks*: mida en blocs del bitmap de blocs.
- *RootInodeNumber*: nombre d'inode del root.
- *FreeInodes*: inodes lliures del sistema de fitxers.
- *FreeBlocks*: blocs lliures del sistema de fitxers.

La inicialització d'algunes d'aquestes variables és diferent respecte a l'ext2. L'ext2 dinàmic tindrà el seu propi identifier, magic, major & minor. Però el que és important és el nombre de sectors, de blocs d'inodes, ocupació dels bitmaps...

En el ext2 el nombre de sectors, blocs i inodes es calculen a partir de la mida de bloc, mida del sector, i nombre de sectors. Però en aquest cas és diferent.

El que volem és utilitzar el mínim espai possible per tal de poder crear un ext2 dinàmic. Els elements bàsics per aquest són: el *Master Boot Record*, el *Superblock*, els inodes especials, els bitmaps, el directori del root, i la llista d'inodes.

Sabent això necessitem només 4 inodes (el quart és el del root).

Per tant necessitem un bloc per:

- Sector de *Boot*
- *Superblock*
- Inode Bitmap Blocs
- Inode Bitmap Inodes
- Inode llista inodes
- Contingut del bitmap de blocs
- Contingut del bitmap d'inodes
- Bloc pel directori *root*
- Bloc per l'inode del *root*

És necessari tenir dos blocs per l'inode i el directori root perquè són creats en el moment de formatejar el sistema de fitxers. Això fa un total de 9 blocs necessaris per crear-lo. El càlcul del nombre de blocs que ocupen els bitmaps es mostra en la següent figura (explicat anteriorment):

```
SB->InodesBitmapSize = (SB->NumberOfInodes / 8)+1;
SB->InodesBMBlocks = 1+(SB->InodesBitmapSize / BlockSize);

SB->BlocksBitmapSize = (SB->NumberOfBlocks / 8)+1;
SB->BlocksBMBlocks = 1+(SB->BlocksBitmapSize / BlockSize);
```

Figura 11. Inicialització mides dels bitmaps

Així doncs i per acabar, el nombre de sectors mínim que especifiquem és de: 9 blocs * (BlockSize/SectorSize) + 1 sector del MBR.

La diferència amb l'ext2 estàtic deixant de banda l'estructura és que aquí els blocs de dades i inodes no es demanen fins que és realment necessari.

7.2 Formateig del sistema de fitxers dinàmic

Prèviament s'ha esmentat que la diferència entre l'estàtic i el dinàmic és la introducció dels 3 inodes especials que gestionaran els bitmap d'inodes, de blocs i la llista d'inodes després de la creació del *Superblock*.

Abans de tot, explicaré ràpidament l'estructura dels inodes que utilitzem per després entendre com es du a terme la seva creació i inicialització.

Com es pot veure en la figura 12, l'inode té diferents paràmetres: nombre d'inode, tipus, referències, mida del fitxer, mida real del fitxer i els punters a blocs de dades.

```
struct Inode
{
    UINT64 Inode_number;                    /* Number of Inode */
    UINT64 type;                            /* Inode type */
    UINT64 references;                      /* Number of references to this Inode */
    UINT64 filesize;                        /* Size of the file */
    UINT64 realfilesize;                    /* Total size allocated for the file */
    UINT64 direct[TOTALINDEXES];            /* Block indexes */
};
```

Figura 12. Estructura de l'inode

A continuació es pot llegir el significat de cadascun d'ells:

- *Inode_number:* és l'identificador únic de l'inode.
- *Type:* indica si es tracta d'un fitxer o un directori

- *References:* nombre de referències al fitxer
- Filesize: és la mida en bytes del fitxer.
- Realfilesize: és el nombre de blocs que ocupa el fitxer.
- Direct: conté tots els punters (directes, indirectes, doble indirectes i triple indirectes en el nostre cas) als blocs de dades del fitxer.

Explicat això ja podem entrar en detall de la creació d'aquests inodes.

Primer de tot cal utilitzar la funció malloc per demanar memòria per aquests. La quantitat a demanar és la mida de bloc del sistema de fitxers. En cas que no es pugui demanar aquesta memòria caldrà alliberar la corresponent a les estructures que ja estiguin creades i retornar un codi d'error. Un cop tenim demanada la memòria pels tres inodes cal inicialitzar-la a 0 amb la funció ZeroMemory i a continuació crear-los. Aquests inodes especials els assigno els números d'inodes 0, 1 i 2 manualment.

Com que ara els bitmaps i la llista d'inodes els tractem com a fitxers, hem d'indicar als inodes especials a quin bloc comencen les seves dades. Per triar el número de bloc he escollit els primers disponibles manualment (5, 6 i 7). Els 5 primers (0, 1, 2, 3 i 4) estan ocupats pel *Boot*, el *Superblock* i els 3 inodes especials respectivament. Després de tenir els inodes creats i inicialitzats, cal marcar-los en el bitmap d'inodes com a utilitzats. L'annex A.1 mostra aquest procediment per l'inode del bitmap de blocs.

A més a més cal indicar a l'inode quant ocupen aquests fitxers. Com s'ha comentat prèviament (i no fa mal recordar-ho) l'inode guarda dos tipus de mida: la del fitxer i la que ocupa en disc.

En el cas de l'inode de la llista d'inodes, sabem que és un fitxer que guarda la informació d'aquests i que ocupen la mida del bloc per tant el *FileSize* i el *RealFileSize* són iguals a la mida de bloc.

En els inodes de bitmaps la cosa canvia. Pel bitmap de blocs, el seu *FileSize* serà de dos bytes perquè és el mínim necessari per a poder crear el sistema de fitxers. En canvi el bitmap d'inodes el *FileSize* és igual a un byte perquè és suficient per representar-ho. La justificació d'això es pot trobar en els detalls de la implementació de la creació del *Superblock*.

Un cop els inodes ja tenen aquesta informació, cal actualitzar el bitmap de blocs indicant quins estan en ús.

En aquest punt només ens falta indicar a l'inode de la llista d'inodes on comença el seu fitxer. Això es fa en el moment de crear el directori root. A més a més de demanar un inode i un bloc pel seu directori com es feia en l'ext2 estàtic, ara ens cal demanar un bloc més en el qual guardarem l'inode.

Quan tenim el número de bloc demanat per a l'inode del root, li indiquem a l'inode de la llista d'inodes que les dades del seu fitxer comencen en aquest bloc. Un cop indicat això, es procedeix a escriure la informació a disc.

En finalitzar la creació del directori root hem d'escriure al *Softdisk* tota la informació dels tres inodes, del *Superblock*, i dels bitmaps.

Per escriure al *Softdisk*, cal passar el número de bloc i les dades a escriure.

El número de bloc serà 2, 3 o 4, que és corresponent als blocs on es troben els tres bitmaps especials i també és necessari escriure a disc els blocs 5 i 6 perquè són els continguts dels bitmaps de blocs i inodes. L'annex A.2 mostra l'escriptura a disc de l'inode del bitmap de blocs.

A conseqüència del canvi d'estructura del sistema de fitxers cal implementar aquelles accions del sistema de fitxers que es veuen afectades per la nova estructura.

A continuació es detalla la implementació d'aquestes funcions.

7. 3 Mount del sistema de fitxers

A l'hora de muntar el sistema de fitxers dinàmic cal llegir els 3 inodes especials així com el *Superblock*. Per fer-ho cal demanar memòria amb malloc amb una mida de bloc de disc. Si no podem demanar memòria, cal alliberar les estructures creades prèviament i retornar un codi d'error.

Un cop tenim la memòria procedim a llegir la informació del disc. Per fer les lectures de disc és necessari indicar quants sectors es volen (mida bloc / mida sector en el nostre cas) i en quin sector comencen les dades (número de bloc * sectors per bloc on número de bloc serà 2, 3, 4 per llegir cadascun dels inodes especials). L'annex A.3 mostra la lectura de l'inode del bitmap de blocs.

Un cop estan carregats a memòria ja tenim el sistema de fitxers muntat.

7.4 Unmount del sistema de fitxers

A diferència del sistema de fitxer estàtic on els bitmaps els escrivia indicant en quin bloc començaven i quants ocupaven, ara cal fer un WriteFile d'aquests per tal que es guardin abans desmuntar el sistema de fitxers.

Abans d'escriure res, cal mirar si el bitmap està carregat a memòria perquè en cas contrari no és necessari realitzar cap escriptura perquè voldrà dir que no hem utilitzat nous blocs o inodes.

Per escriure'ls cal indicar quin és l'inode del fitxer, què volem escriure, quin offset volem i quina és la mida de les dades.

La mida de les dades a escriure és el nombre de bytes que ocupa el bitmap corresponent i el offset és 0 perquè pot ser que s'alliberin fitxers del principi i així ens assegurem que estem escrivint tot el bitmap correctament. Un cop hem escrit el bitmap i no hem rebut cap error, alliberem la memòria que ocupava i el posem a NULL. Tot això es pot veure en l'annex A.4.

Cal afegir també l'escriptura a disc dels inodes especials. Aquestes escriptures es duen a terme igual que quan fem un format del sistema de fitxers (explicat més amunt).

7.2 Inodes

En aquest apartat s'explica com es realitzen les accions de llegir i escriure els inodes així com l'obertura dels fitxers.

7.2.1 Llegir els inodes

Aquesta funció rep dos paràmetres: el primer indica quin inode volem llegir i el segon on guardarem la informació d'aquest.

En el sistema de fitxers estàtic fèiem un ReadBlock de la cache passant-li el número de bloc on estava l'inode perquè podiem saber-ho. Ara a l'hora de llegir els inodes la cosa canvia perquè (exceptuant els inodes especials) aquests estan guardats en un fitxer i no sabem en quins blocs estan.

Per tant es distingeixen dos casos: inode especial o no.

- Inode especial

  Si el número d'inode és inferior a tres, vol dir que volem llegir els inodes dels bitmaps o de la llista d'inodes. Per llegir-los, simplement cal passar com a paràmetres el número de bloc on es troba l'inode especial i el buffer on emmagatzemem les dades. El número de bloc és el número d'inode + 2 perquè els dos primers blocs corresponent al *Boot* i al *Superblock*.

- Inode no especial

  Com he comentat, ara aquests inodes estan en un fitxer i per tant hem de llegir-los d'allà. Per fer-ho caldrà fer una lectura al fitxer de la llista d'inodes passant-li l'inode especial, el buffer on es guarda la informació, el offset corresponent, i el nombre de bytes a llegir que es correspon amb la mida de bloc.
  El offset és igual a Mida bloc * (número d'inode - 3) perquè els inodes ocupen igual que un bloc i cal restar 3 al número d'inode perquè els tres primers corresponen als especials i no es troben en el fitxer amb la resta. És a dir, a l'hora de llegir per exemple l'inode número 3 que és el primer de la llista, el seu offset serà de zero.

L'annex A.5 mostra la implementació de la lectura dels inodes.

7.2.2 Escriure els inodes

Aquesta funció rep dos paràmetres: el *inode_number* de l'inode que volem escriure i el *buffer* amb tota la informació de l'inode.

Com en el ReadInode, hem de distingir dos casos:

- Inode especial

  Si el número d'inode és inferior a tres, vol dir que volem escriure els inodes dels bitmaps o de la llista d'inodes. Per escriure'ls, simplement cal passar com a paràmetres el número de bloc on es troba l'inode especial i el buffer on tenim les dades. El número de bloc és el número d'inode + 2 perquè els dos primers blocs corresponent al *Boot* i al *Superblock*.

- Inode no especial

  Com he comentat, ara aquests inodes estan en un fitxer i per tant hem d'escriure'ls allà. Per fer-ho caldrà fer una escriptura al fitxer de la llista d'inodes passant-li l'inode especial, el buffer on es guarda la informació, el offset corresponent, i el nombre de bytes a escriure que es correspon amb la mida de bloc.
  El offset és igual a: mida bloc * (número d'inode - 3) perquè els inodes ocupen igual que un bloc i cal restar 3 al número d'inode perquè els tres primers corresponen als especials i no es troben en el fitxer amb la resta. És a dir, a l'hora d'escriure per exemple l'inode número 3 que és el primer de la llista, el seu offset serà de zero.

L'annex A.6 mostra la implementació de l'escriptura dels inodes.

7.2.3 Buscar inodes lliures

En aquesta funció el sistema de fitxers estàtic només s'encarregava de retornar el número d'inode lliure o que no quedaven. En el dinàmic realitza més coses que es detallen a continuació.

Primer de tot, cal comprovar si tenim el bitmap en memòria. Si no és el cas, cal demanar memòria i carregar-lo per poder treballar amb ell a continuació i en les pròximes vegades que accedim per demanar un inode.

Com que ara és un fitxer, necessitem fer una lectura d'aquest passant com a paràmetres l'inode del bitmap d'inodes, el *buffer* on guardarem la informació, l'offset i quants bytes volem llegir.

La mida a llegir és igual a la quantitat de bytes que ocupa el bitmap d'inodes i que sabem perquè aquesta informació està emmagatzemada en el *Superblock*.

A continuació cal comprovar si tenim inodes lliures. Si en tenim, simplement cridem a la funció *getFirstFreeFrom* que donat un bitmap (bitmap d'inodes en aquest cas), una posició d'inici i el nombre d'inodes totals, retorna el número d'inode lliure.

En cas que no en quedin, cal realitzar un seguit d'accions:

Com que el bitmap el tenim carregat a memòria i les modificacions no estan a disc, cal fer un WriteFile del bitmap i un cop fet alliberem la memòria que ocupava, demanem memòria nova pel bitmap i l'inicialitzem a zero (valors dels inodes lliures).

A continuació escrivim en el bitmap d'inodes, on tenim tota la informació dels inodes, aquest bitmap inicialitzat a zero. En aquest cas l'offset és la mida en bytes que ocupa el bitmap d'inodes i els bytes a escriure depenen de la configuració establerta en el moment d'implementar el codi per l'administrador.

Incrementar el bitmap i demanar inodes comporta mantenir actualitzats camps del *Superblock*: inodes lliures, nombre d'inodes, mida en bytes del bitmap d'inodes i els blocs que ocupa el bitmap. Aquests camps es veuran incrementats de la següent forma:

- Nombre d'inodes es veurà incrementat en 8 * nombre de bytes demanats.
- Inodes lliure es veurà incrementat en 8 * nombre de bytes demanats.
- Mida en bytes del bitmap: (Nombre d'inodes / 8) + 1.
- Mida en blocs del bitmap: (Mida en bytes del bitmap / Mida Bloc) + 1.

Després de tot això, ja tenim el *Superblock* actualitzat i abans de demanar quin inode està lliure cal carregar a memòria el bitmap per poder accedir a ell. És important fer aquesta lectura després d'actualitzar les variables del *Superblock* perquè sinó estaríem llegint una quantitat de bytes diferent de la real.

Un cop ja tenim tot actualitzat procedim a demanar el primer inode lliure passant el bitmap d'inodes, la quantitat d'inodes i on volem guardar el número. Si obtenim un resultat correcte, cal indicar en el bitmap que l'inode està ocupat i reduir el nombre d'inodes lliures. Altrament, es retorna un error indicant que no en queden. Tota la implementació d'aquesta funció es pot veure en l'annex A.7.

7.2.4 Alliberar inodes

Aquesta funció rep com a paràmetre un número d'inode. Per alliberar-lo, simplement caldrà modificar el bitmap d'inodes assignant zero a la posició que indica el número d'inode. Abans però, caldrà comprovar si tenim el bitmap a memòria i, si no el tenim, haurem de demanar una quantitat de memòria equivalent a la mida d'un bloc, llegir-lo passant-li l'inode del bitmap d'inodes, el buffer on volem guardar la informació i on acabem de guardar la memòria demanada, i l'offset que és zero perquè el volem llegir sencer.

Si tant la lectura com la petició de memòria no donen errors, actualitzem la variable del *Superblock* que indica la quantitat de inodes lliures i actualitzem el bitmap, com s'ha explicat a l'inici d'aquesta secció. L'annex A.8 mostra el procediment.

7.2.5 Obrir fitxers

En el sistema de fitxers estàtic, a l'hora d'obrir un fitxer es busca l'inode i després realitza un pin del bloc de l'inode. Això es pot fer perquè, com sabem on comencen i acaben els blocs dels inodes, podem calcular amb el número d'inode quin és el bloc on està. Això ara no podem fer-ho perquè, en estar en un fitxer la informació dels inodes, no tenim manera de saber el número de bloc on està.

Per tant la funció del sistema de fitxers d'obrir ja no realitza el pin del bloc, simplement retorna el punter de l'inode que és i que tenim guardat a la Buffer Cache.

La modificació es realitza a nivell del VirtualFileSystem i simplement cal replicar l'inode a la taula d'inodes en lloc de guardar aquest punter en la taula.

7.3 Blocs

En aquest apartat s'explica com es realitza la gestió dels blocs del sistema de fitxers dinàmic.

7.3.1 Buscar blocs lliures

Primer de tot, cal comprovar si tenim el bitmap en memòria. Si no és el cas, cal demanar memòria i carregar-lo per poder treballar amb ell a continuació i en les pròximes vegades que accedim per demanar un bloc.

Com que ara és un fitxer, necessitem fer una lectura d'aquest passant com a paràmetres l'inode del bitmap de blocs, el *buffer* on guardarem la informació, l'offset i quants bytes volem llegir.

La mida a llegir és igual a la quantitat de bytes que ocupa el bitmap de blocs i que sabem perquè aquesta informació està emmagatzemada en el *Superblock*.

A continuació cal comprovar si tenim blocs lliures. Si en tenim, simplement cridem a la funció *getFirstFreeFrom* que donat un bitmap (bitmap de blocs en aquest cas), una posició d'inici i el nombre de blocs totals, retorna el número de bloc lliure.

En cas que no en quedin, cal realitzar un seguit d'accions:

Com que el bitmap el tenim carregat a memòria i les modificacions no estan a disc, cal fer un WriteFile del bitmap i alliberem la memòria que ocupava.

Un cop realitzat, el sistema de fitxers crida a una funció del *Softdisk* per tal de demanar més blocs. A aquesta funció cal passar-li dos paràmetres que es corresponen amb el nombre de blocs que es vol fer créixer el sistema de fitxers i la mida de bloc. Aquesta funció s'explica més endavant.

Si aquesta funció retorna que tot ha anat correcte, cal actualitzar el nombre de sectors del *Softdisk*. Aquesta informació està guardada al *Superblock* per tant cal multiplicar el nombre de blocs demanats per sectors per bloc i sumar el resultat al nombre de sectors actuals.

Un cop tenim el *Softdisk* incrementat, cal incrementar el bitmap de blocs. Per fer-ho primer de tot demanem memòria (tanta com bytes volem incrementar el sistema de fitxers) i la inicialitzem tota a zero (valor dels blocs lliures).

A continuació escrivim en el bitmap de blocs que tenim al disc (on tenim tota la informació dels blocs) aquest bitmap inicialitzat a zero. En aquest cas l'offset és igual al nombre de bytes que ocupa el bitmap i els bytes a escriure depenen de la configuració establerta en el moment d'implementar el codi per l'administrador

Incrementar el bitmap i demanar blocs comporta mantenir actualitzats camps del *Superblock*: blocs lliures, nombre de blocs, mida en bytes del bitmap de blocs i els blocs que ocupa el bitmap.

- Nombre de blocs es veurà incrementat en 8 * nombre de bytes demanats.
- Blocs lliures es veurà incrementat en 8 * nombre de bytes demanats.
- Mida en bytes del bitmap: (Nombre de blocs / 8) + 1.
- Mida en blocs del bitmap: (Mida en bytes del bitmap / Mida Bloc) + 1.

Després de tot això, ja tenim tot actualitzat i abans de demanar quin bloc està lliure cal carregar a memòria el bitmap per poder accedir a ell. És important fer aquesta lectura després d'actualitzar les variables del *Superblock* perquè si no estaríem llegint una quantitat de bytes diferent de la real.

A continuació procedim a demanar el primer bloc lliure passant a la funció el bitmap de blocs, la quantitat de blocs i on volem guardar el número. Si obtenim un resultat correcte, cal indicar en el bitmap que el bloc està ocupat i reduir el nombre de blocs

lliures. Altrament, es retorna un error indicant que no en queden. Tota la implementació d'aquesta funció es pot veure en l'annex A.9.

7.3.2 Alliberar blocs

Aquesta funció rep com a paràmetre un número de bloc. Per alliberar-lo, simplement caldrà modificar el bitmap de blocs assignant zero a la posició que indica el número de bloc. Abans però, caldrà comprovar si tenim el bitmap a memòria i, si no el tenim, haurem de demanar una quantitat de memòria equivalent a la mida d'un bloc, llegir-lo passant-li l'inode del bitmap de blocs, el buffer on volem guardar la informació i on acabem de guardar la memòria demanada, i l'offset que és zero perquè el volem llegir sencer.

Si tant la lectura com la petició de memòria no donen errors, actualitzem la variable del *Superblock* que indica la quantitat de blocs lliures i actualitzem el bitmap, com s'ha explicat a l'inici d'aquesta secció. L'annex A.10 mostra la implementació d'aquest procediment.

7.3.3 Ampliar el nombre de blocs

La implementació d'aquesta funció ens proporciona l'augment que desitgem en la mida del sistema de fitxers. A continuació l'explico en detall.

Aquesta funció rep dos paràmetres: mida del bloc i nombre de blocs a demanar.

El primer que cal fer és calcular quants bytes incrementem el fitxer. Aquesta quantitat es calcula multiplicant la mida de bloc pel nombre de blocs a créixer.

Un cop sabem això cal fer dues coses: moure el punter del fitxer la distància calculada a partir del seu final i indicar on acaba el fitxer. Per realitzar aquestes accions s'utilitzen dues funcions:

- La funció SetFilePointer és la que ens permet moure el punter del fitxer la distància calculada a partir del final del fitxer. Aquesta funció necessita un handle del fitxer, la distància a moure el punter i des d'on ha de moure'l.
- La funció SetEndOfFile ens permet indicar el final del fitxer després de fer-lo créixer. Només necessita el handle del fixter.

Després d'aconseguir tenir més blocs en el sistema de fitxers, cal actualitzar el nombre de sectors que emmagatzema el *Softdisk* i el *MBR*. Primer calculem el nombre de sectors que hem incrementat el sistema de fitxers dividint el nombre de bytes incrementats (nombre de blocs * mida bloc) entre la mida d'un sector. El resultat el sumem al nombre de sectors.

Abans de finalitzar cal actualitzar el MBR i escriure'l a disc. Per fer això actualitzem el MBR dient-li que el nombre de sectors és igual a la suma dels que teníem més els calculats. A continuació movem el punter del fitxer a l'inici d'aquest amb la funció SetFilePointer i escrivim el MBR en el bloc 0 i només un sector (el que ocupa el MBR).

Aquest pas és importantíssim perquè, si no el realitzem, quan tornem a muntar el sistema de fitxers i ha crescut dinàmicament, estaríem amb un nombre de sectors erroni provocant que el sistema fallés. En l'annex A.11 es pot trobar la implementació d'ampliació del nombre de blocs.

7.4 Integració en l'estructura general

Un cop tenim tot el sistema de fitxers dinàmic implementat, cal integrar-lo a l'estructura general. Per fer-ho és necessari modificar algunes de les funcions del VirtualFileSystem/FileSystem. A continuació s'expliquen aquestes modificacions.

7.4.1 Format en el VirtualFileSystem

Quan es crea el *Softdisk*, just després el VirtualFileSystem realitza un format del sistema de fitxers que s'ha especificat, passant-li la mida de bloc i el *Softdisk* per poder escriure a disc. Per tant ens cal afegir el cas del nou sistema de fitxer perquè el tingui en compte.

Aquest nou cas serà igual que el del sistema de fitxers ext2 i realitzarà les següents accions:

- Creació del sistema de fitxers dinàmic
- Format del sistema de fitxers dinàmic

Un cop realitzades retornarà el resultat (Annex A.12).

7.4.2 Mount en el VirtualFileSystem

A l'hora de muntar el sistema de fitxers, el VirtualFileSystem admet dos tipus de paràmetres per decidir quin sistema de fitxers està sent utilitzat. El primer és indicar-li directament amb el nom i el segon és indicar-li que el detecti ell.

En la primera situació, s'ha hagut d'afegir el cas del nou sistema de fitxers. Un cop sap que és aquest, procedeix a realitzar les següents funcions:

- Creació del sistema de fitxers dinàmic
- Mount del sistema de fitxers dinàmic.

Si tot va bé, el sistema seguirà executant-se. Altrament retornarà un codi d'error.

En canvi, si al VirtualFileSystem se li indica que detecti el tipus de sistema de fitxers (FS_AUTO), cal afegir les modificacions en la funció de detectar quin és. En l'annex A.13 es pot veure la implementació d'aquests dos casos.

Les modificacions que cal realitzar en la funció detect (Annex A.14) consisteixen a crear una instància del sistema de fitxers nou i realitzar un detect d'aquest. La funció detect el que fa és llegir el *Superblock* que tenim en disc i comparar les dades que identifiquen el sistema de fitxers del disc amb les dades generals del sistema de fitxers. Si tot va correcte, un cop ha detectat quin sistema de fitxers tenim es procedeix a muntar-lo.

8. Testing

El testing del sistema ens serveix per comprovar que funciona tal com esperem. A l'inici d'aquest projecte es van especificar un seguit de requeriments que ha de complir el sistema de fitxers:

- Ha de créixer dinàmicament
- Robust enfront de possibles inconvenients
- Eficient en espai.

S'ha dissenyat un test per comprovar i demostrar que funciona bé i que realment compleix amb els requeriments esmentats.

El test, que a continuació explicaré, s'ha realitzat amb 1.000 fitxers, amb una mida de bloc de 4KB i una mida de sector de disc de 512B. Tot i que la mida de bloc i de sector poden ser diversos, només es realitzen les proves amb aquest dos. Això és perquè en aquest projecte no he implementat l'ext2, sinó que l'he ampliat per dinamitzar el nombre de blocs.

8.1 Test realitzat

En aquest text s'han realitzat proves per tal de comprovar que les operacions read, write, open, close i delete sobre els fitxers funcionen bé amb la nova estructura del sistema de fitxers dinàmic.

Durant aquestes operacions, s'ha anat parant l'execució per poder demostrar la robustesa del sistema mirant que la gestió dels bitmaps sigui correcte, així com la dels inodes, entre altres coses.

El primer que fem és crear el *Softdisk* i muntar-lo. Per fer això, indiquem que volem que les mides siguin 4K mida de bloc, 512B mida del sector i el sistema de fitxers FS_DYNAMIC_EXT2. El nombre de sectors si es tracta d'aquest sistema de fitxers dinàmic, s'ignora.

Un cop creat el sistema de fitxers, podem observar (figura 13) que el nombre de blocs són 9 i el nombre d'inodes són 4. Per tant veiem que el sistema de fitxers comença amb el mínim necessari tal com s'havia explicat prèviament i que no disposa de més blocs ni inodes lliures.

NumberOfBlocks	9
NumberOfSectors	73
BlockSize	4096
SectorsperBlock	8
NumberOfInodes	4
InodeSize	4096
FirstInodeBlock	7
FirstDataBlock	7
InodesBitmapSize	1
InodesBMBlocks	1
BlocksBitmapSize	2
BlocksBMBlocks	1
RootInodeNumber	3
FreeInodes	0
FreeBlocks	0

Figura 13. Informació *Superblock* al crear el sistema de fitxers dinàmic

Ara procedim a crear els fitxers. A l'hora de crear-los, el nom que se'ls dóna és el text "fitxerX.txt" on X és el número de fitxer en ordre de creació.

Per controlar els errors, després de cada crida a la funció CreateFile es comprova si el codi retornat per la funció és un error o no. Si el fitxer no està creat i ha anat tot bé ens informa del succés (figura 14).

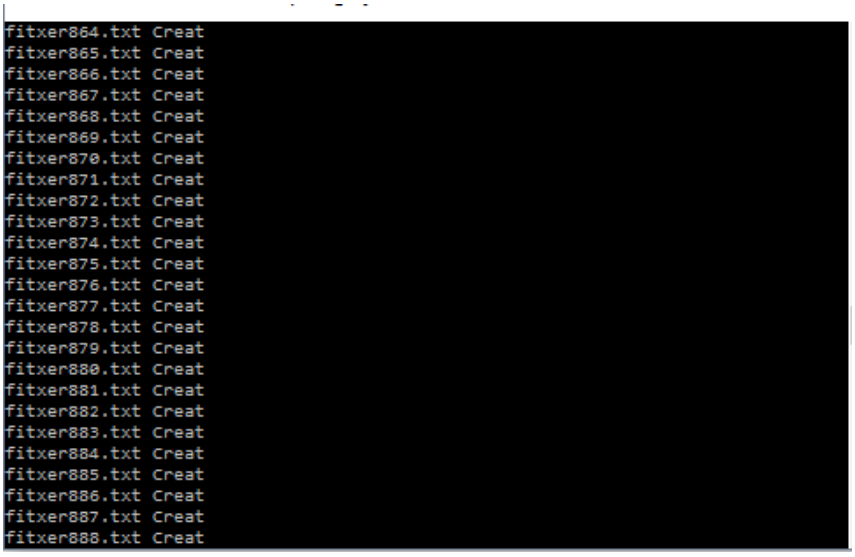


Figura 14. Resultats en consola de la creació dels fitxers

Amb aquesta acció ja estem fent que el sistema de fitxers creixi, però cal mirar que totes les variables quedin actualitzades (figura 15) així com que els bitmaps incrementin (figures 16 i 17). Per veure que la gestió dels bitmaps és correcte, VisualStudio permet visualitzar la memòria utilitzada i els seus valors.

NumberOfBlocks	1041
NumberOfSectors	8329
BlockSize	4096
SectorsperBlock	8
NumberOfInodes	1012
InodeSize	4096
FirstInodeBlock	7
FirstDataBlock	7
InodesBitmapSize	127
InodesBMBlocks	1
BlocksBitmapSize	131
BlocksBMBlocks	1
RootInodeNumber	3
FreeInodes	7
FreeBlocks	14

Figura 15. Informació *Superblock* després de crear 1.000 fitxers

```
0x00FB9918  ff 01 fd fd fd fd dd dd f6
0x00FB9957  00 54 00 42 00 41 00 53 00
```

Figura 16. Contingut bitmap blocs abans de crear fitxers

```
0x00FC0218  ff ff ff ff ff ff ff ff ff
0x00FC0257  ff ff ff ff ff ff ff ff ff
```

Figura 17. Contingut d'una part del bitmap blocs després de crear fitxers

En la figura 16, es pot apreciar que els primers valors són ff 01 que són 16 bits dels quals 9 tenen valor 1. Per tant és correcte perquè en muntar el sistema de fitxers des de zero només tenim 9 blocs ocupats. A la figura 17 es mostra una part del contingut del bitmap després de la creació dels 1.000 fitxers. Veient això podem afirmar que la gestió dels bitmaps s'està realitzant correctament. La posició de memòria és diferent perquè cada cop que ampliem el bitmap s'allibera la memòria que utilitzava.

Les figures 18 i 19 mostren el contingut del bitmap d'inodes abans i després de crear els fitxers.

```
0x01243CF8  0f fd fd fd fd dd dd dd db
0x01243D37  00 33 00 32 00 2e 00 64 00
```

Figura 18. Contingut bitmap inodes abans de crear fitxers

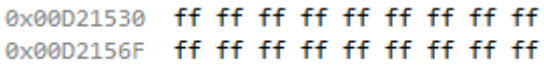


Figura 19. Contingut d'una part del bitmap inodes després de crear fitxers

Hem vist que la gestió dels bitmaps és correcte però cal assegurar-nos que el número de bytes que es demana cada cop són els que realment creix el bitmap. Per demostrar això s'ha pres una imatge de la memòria del bitmap de blocs i el bitmap d'inodes just després d'incrementar el sistema de fitxers per primer cop així com el nou valor de les variables del *Superblock*.

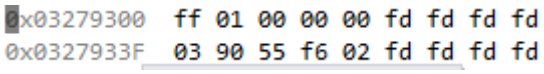


Figura 20. Contingut bitmap blocs després de demanar-ne

En la figura 20 es pot apreciar, en comparació amb la figura 18, que 3 bytes han passat a estar inicialitzats a zero. Actualment el sistema de fitxers demana 24 blocs o 24 inodes (es pot canviar modificant el codi) i per això només s'ha fet créixer el fitxer en 3 bytes.

La figura 21 mostra el mateix increment però pel bitmap d'inodes.

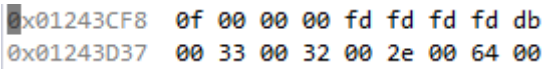


Figura 21. Contingut bitmap inodes després de demanar-ne

Amb l'increment de blocs i inodes, cal veure si en el *Superblock* la informació pertinent s'ha actualitzat com hauria. La figura 22 mostra la informació del *Superblock* després de fer créixer tant el nombre de blocs com d'inodes.

NumberOfBlocks	33
NumberOfSectors	265
BlockSize	4096
SectorsperBlock	8
NumberOfInodes	28
InodeSize	4096
FirstInodeBlock	7
FirstDataBlock	7
InodesBitmapSize	4
InodesBMBlocks	1
BlocksBitmapSize	5
BlocksBMBlocks	1
RootInodeNumber	3
FreeInodes	23
FreeBlocks	24

Figura 22. *Superblock* després de demanar 24 blocs i inodes

Si ens fixem, el nombre de blocs ha crescut 24 unitats (teniem 9) i el nombre de sectors ha crescut 192 (73 que teníem més 192 nous = 265 sectors).

A més a més, veiem que la variable que conté la mida en bytes del bitmap de blocs ha passat a ser 5 bytes (teniem 2 bytes i l'hem incrementat en 3) i que el nombre de blocs lliures és 24.

Tambe es pot apreciar l'increment d'inodes (això és així perquè s'ha pres la imatge en el moment de crear un directori nou i primer es demana l'inode).

Veiem que el nombre d'inodes s'ha incrementat en 24 (teniem 4) i que ara ens diu que tenim 23 lliures (perquè un l'hem agafat per crear el directori per poder agafar el valor de les variables). A més a més el bitmap s'ha incrementat en 3 bytes corresponents als demanats (només ocupava 1 byte).

Després de comprobar que la gestió de bitmaps és correcte quan afegim blocs o inodes, cal veure si aquests es modifiquen bé quan esborrem un fitxer.

Per provar-ho, s'ha escollit un dels fitxers per esborrar. Abans d'esborrar el fitxer, consultem la informació del seu inode:

◢ inode		0x04869798 {Inode_number=95 type=1 references=1 ...}
	Inode_number	95
	type	1
	references	1
	filesize	126
	realfilesize	4096
▷	direct	0x048697c0 {304, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
inode_number		95

Figura 23. Informació de l'inode 95

Veiem que el seu inode_number és el 95 i que les seves dades comencen en el bloc 304.

Quan esborrem el fitxer, aquest inode i bloc queden alliberats i això es veu reflectit en els bitmaps (figures 24 i 25).

```
Dirección: 0x01398B90
0x01398B90  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff fe ff ff ff ff ff
```

Figura 24. Bloc alliberat en vermell

```
Dirección: 0x042B5700
0x042B5700  ff ff ff ff ff ff ff ff ff ff ff 7f ff
```

Figura 25. Inode alliberat (7f)

Un cop veiem que els bitmaps són correctes procedim a tornar a crear el fitxer amb el mateix nom i els canvis que es produeixen en els bitmaps són els següents:

```
Dirección: 0x01398B90
0x01398B90  ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
```

Figura 26. Bitmap de blocs al crear un altre cop el fitxer

```
Dirección: 0x042B5700
0x042B5700  ff ff ff ff ff ff ff ff ff ff ff ff ff
```

Figura 27. Inode de blocs al crear un altre cop el fitxer

Com es pot apreciar, en el cas del bitmap de blocs on abans hi havia "fe" ara trobem "ff" amb la qual cosa podem afirmar que el bloc alliberat a l'esborrar el fitxer s'ha tornat a agafar (bloc 304).

En la segona figura (27) es mostra el bitmap d'inodes i que on hi havia "7f" ha passat a "ff" agafant l'inode 95.

Per acabar de demostrar que funciona bé, s'ha escrit en el fitxer una mica més d'un bloc per veure si realment aquesta informació es veu registrada i coincideix:

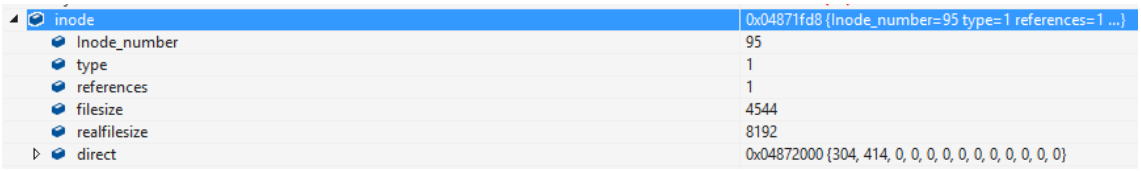


Figura 28. Informació de l'inode del nou fitxer

Efectivament tenim que l'inode_number del fitxer és 95 i que les seves dades comencen al bloc 304.

Un cop ens hem assegurat que la gestió de bitmaps és correcte, passem a comprovar que podem escriure en tots els fitxers.

Per poder escriure o llegir d'un fitxer necessitem un handle per cadascún d'ells. Per això s'ha declarat un vector global de 1.000 posicions on, en la posició X, tenim el handle del fitxer "fitxerX.txt".

El mateix cas que quan creem els fitxers, per a cada iteració hem de generar el nom que serà el que passem a la funció OpenFile juntament amb els permisos i el handle corresponent. En cas que es produeixi algun error, aquest és escrit a consola i retornem l'error. En cas de succés es mostra la frase "Obrint el fitxer "fitxerX.txt"" com mostra la figura 29.

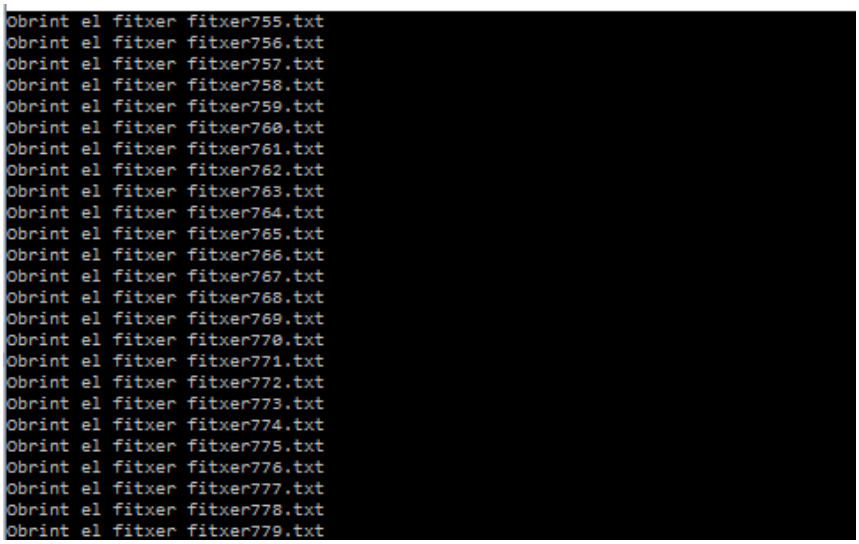


Figura 29. Resultats en consola al obrir els fitxers

Poc després de començar a obrir els fitxers, he parat el programa per comprovar realment que els inodes de fitxers s'estan llegint bé del fitxer on està la llista d'inodes. La figura 30 mostra el fitxer que estem llegint el seu inode i que encara no tenim la seva informació.

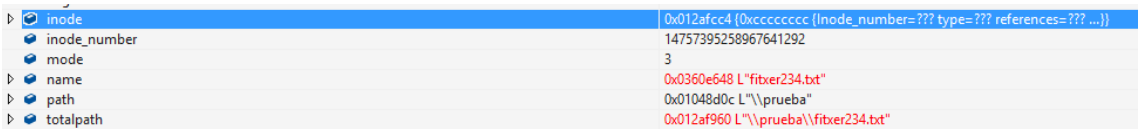


Figura 30. Dades del fitxer "fitxer234.txt"

La següent figura mostra el número d'inode que ha llegit del fitxer on es guarden tots i es correspon amb el buscat:

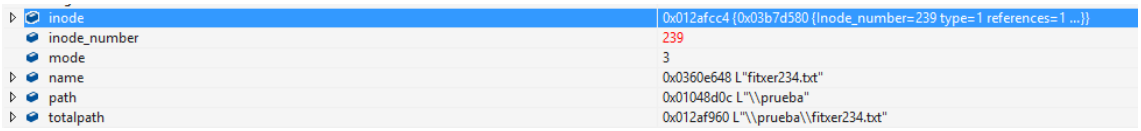


Figura 31. Inode del fitxer "fitxer234.txt"

Veiem que el número d'inode és 239 i no 234. Això és a causa dels inodes creats al formatejar el sistema de fitxers (inode root i els 3 especials) i a l'inode del directori "prueba". Per tant podem afirmar que a l'hora de llegir els inodes del fitxer, l'offset que s'explica en l'apartat corresponent a la implementació de la lectura d'inodes, està ben calculat.

Un cop oberts els fitxers el que fem és escriure en tots ells. La frase a escriure consisteix en "Soc el fitxer numero X" per després identificar el fitxer que estem llegint.

Per escriure cal passar el handle del fitxer, la frase, i la mida d'aquesta. Si no hi ha cap problema, ens ho indica per consola (figura 32). Altrament retorna un error.

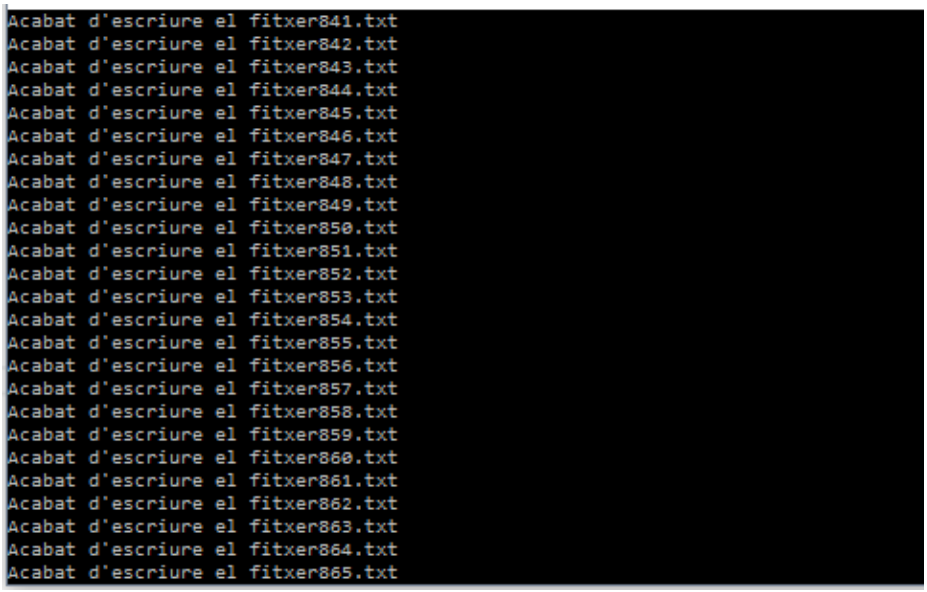


Figura 32. Resultats en consola en escriure els fitxers

El següent pas és veure que les dades just escrites es poden llegir sense cap inconvenient i, per fer això, necessitem posar els punters dels fitxers a l'inici d'aquest amb la funció SetFilePointer.

Després d'això procedim llegint els fitxers i obtenim els resultats que es veuen a la consola (figura 33). Si no pot llegir algun fitxer, la funció retorna error.

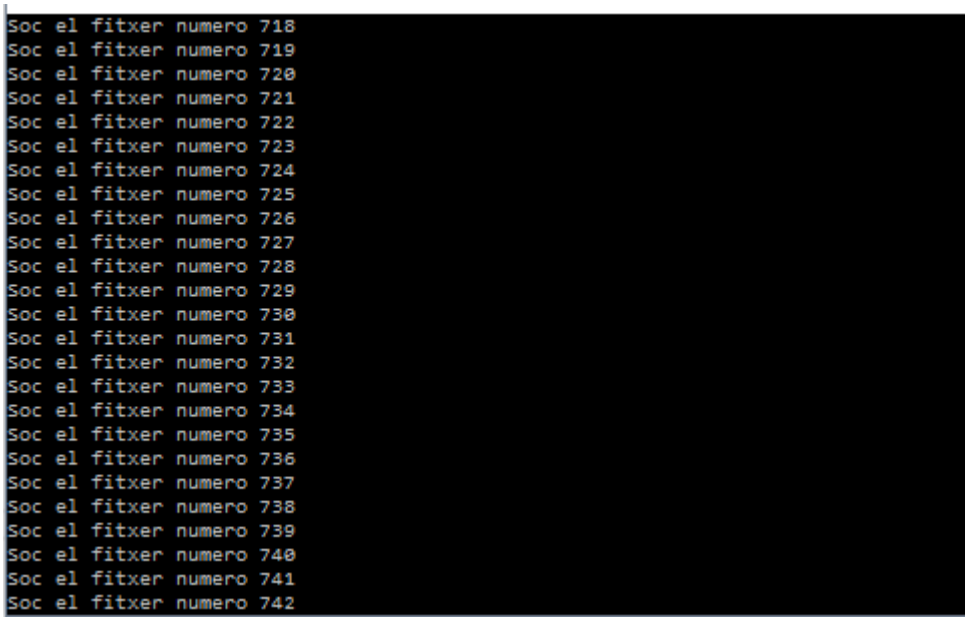


Figura 33. Procés de lectura dels fitxers

Una prova a realitzar un cop tenim creats i escrits tots els fitxers és desmuntar el ext2 dinàmic i tornar-lo a muntar. L'objectiu d'això és comprovar que les dades no es perden mantenint així la consistència.

Per fer això el primer que fem és desmuntar el sistema de fitxers i tornar a muntar-lo. A continuació obrim els fitxers i els llegim un altre cop. A més a més, abans de fer tot això s'han escrit a la meitat dels fitxers una frase al final d'aquests per veure que els resultats són diferents de la lectura prèvia (figura 34).

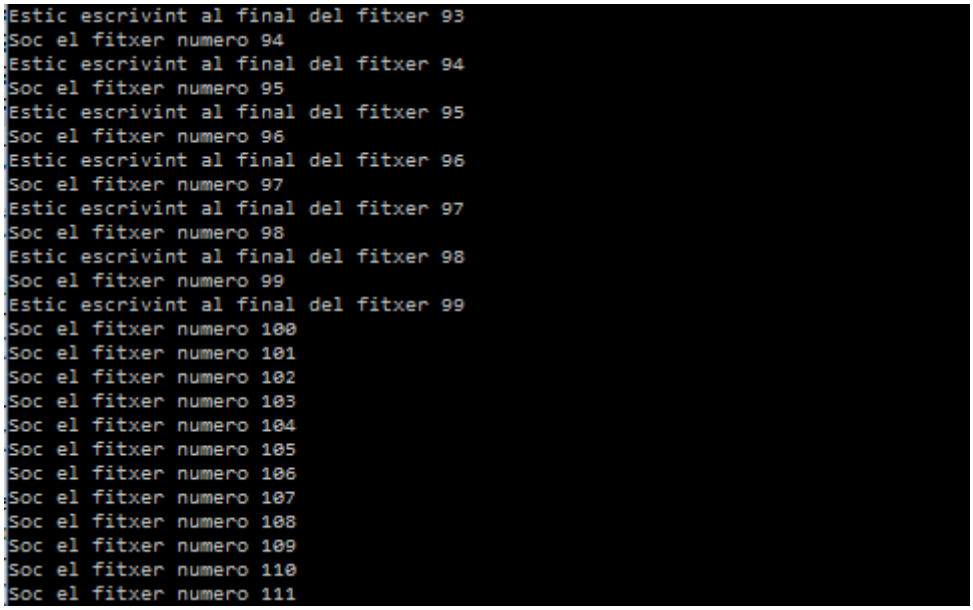


Figura 34. Procés de lectura després de desmuntar i muntar el sistema de fitxers

Per acabar, cal tancar tots els fitxers i veure que no dóna cap error. Per tancar-los, simplement cal indicar el handle de cada fitxer que els tenim guardats en el vector de handles. Si el procés va bé ho indica per consola (figura 35) i si no retorna un error.

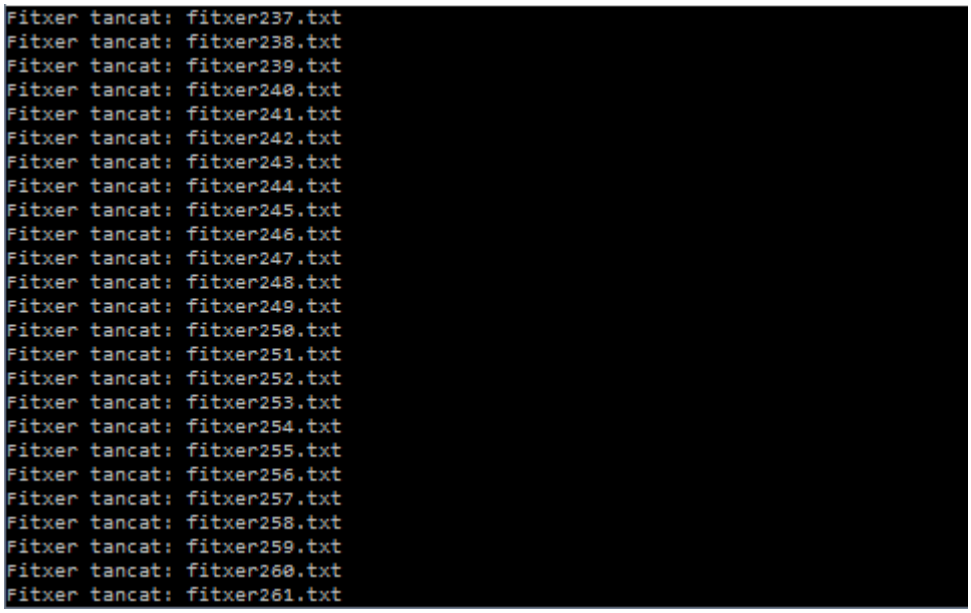


Figura 35. Procés de tancar fitxers

En l'última prova, s'han agafat un parell de fitxers en els quals hem escrit 800 blocs i alguns bytes en cadascun d'ells per comprovar que no dóna cap error el fet d'escriure molta informació en els fitxers. Un cop els hem escrit, procedim a llegir-los (figura 36).

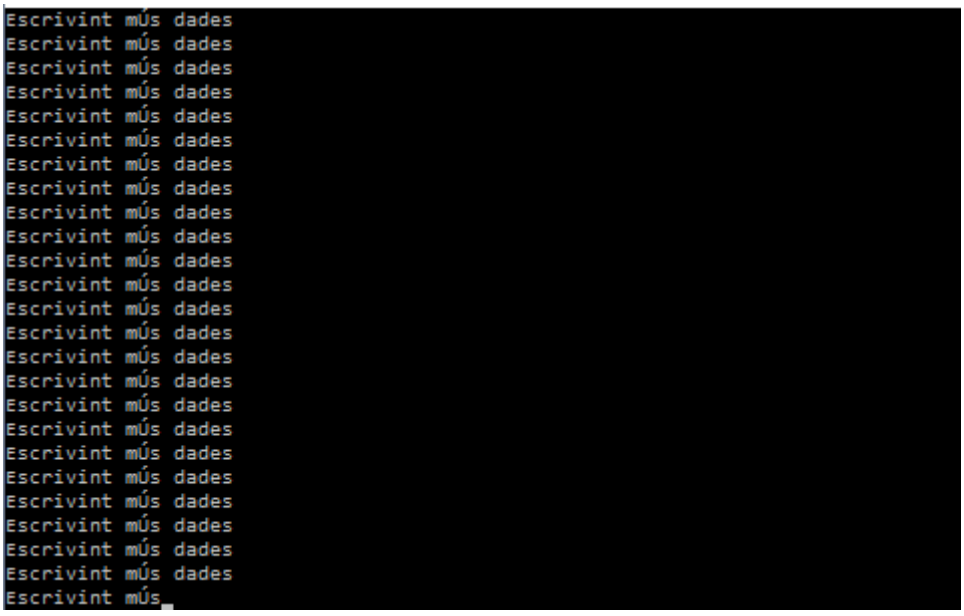


Figura 36. Dades d'un fitxer de 800 blocs

A continuació, es mostren les dades de l'inode d'un d'aquests fitxers:

Figura 37. Inode fitxer gran

Com es pot veure el número de bytes que té el fitxer són 3.276.840 que equival a 801 blocs (bytes/4096) i el realfilesize són els bytes que ocupa realment que són 3.280.896 que equival a 801 blocs. Per tant veiem que la informació de l'inode és correcte.

Amb tot això hem comprovat que el sistema de fitxers funcioni correctament amb la nova estructura interna i que la part estesa del sistema de fitxers sigui robusta.

8.2 Eficiència

Per buscar el percentatge d'eficiència del nostre sistema de fitxers, cal trobar la relació mida dades usuari / mida del *softdisk*.

Per calcular-la partirem de les dades que es mostren en la següent figura:

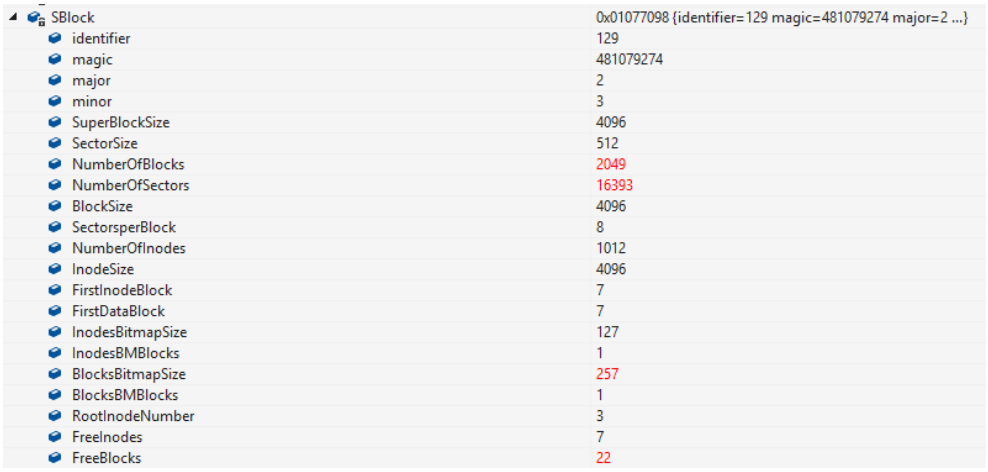


Figura 38. *Superblock* del sistema de fitxers

Aquesta captura ha estat feta després de crear 1.000 fitxers i escriure en ells un bloc de dades.

Aquí podem veure tota la informació del sistema de fitxers en el *Superblock* i les dades que ens interessen són el nombre de blocs i la mida d'aquests.

Amb aquesta informació necessitem trobar quants blocs són dades de l'usuari. Per fer-ho cal saber quants no ho són. En primer lloc sabem que el nostre sistema de fitxers

en el moment de creació només ocupa 9 blocs. A l'hora de crear fitxers, aquests necessiten un inode en el qual es guarda la seva informació. Els inodes no són dades d'usuari perquè formen part de les estructures que necessita el sistema de fitxers per gestionar-se. Dit això, al nombre de blocs cal restar-li:

- 9 blocs inicials.
- 1.000 blocs dels inodes dels fitxers.
- 22 blocs que estan lliures.

Això fa un total de 1.018 blocs que són 4.169.728 bytes (1.018 blocs * 4.096 bytes/bloc).

Per altra banda, la mida total en bytes del *Softdisk* és 8.392.704 bytes (2.049 blocs * 4.096 bytes/bloc).

Un cop tenim que ocupa cada cosa trobem que l'eficiència en espai és la següent:

Eficiència en espai = dades usuari en bytes / mida *Softdisk* en bytes = 4.169.728 bytes / 8.392.704 bytes = 0.5 * 100 = 50 % d'eficiència.

Cal tenir en compte que per aquest càlcul els bitmaps només ocupen 2 blocs entre els dos.

Acabem de veure un cas on tenim molts fitxers amb un bloc de dades cadascun d'ells, però que passa si en tenim pocs i amb moltes dades?

La següent figura mostra el *Superblock* del nou cas:

SBlock	0x015f6180 {identifier=129 magic=481079274 major=2 ...}
identifier	129
magic	481079274
major	2
minor	3
SuperBlockSize	4096
SectorSize	512
NumberOfBlocks	1641
NumberOfSectors	13129
BlockSize	4096
SectorsperBlock	8
NumberOfInodes	28
InodeSize	4096
FirstInodeBlock	7
FirstDataBlock	7
InodesBitmapSize	4
InodesBMBlocks	1
BlocksBitmapSize	206
BlocksBMBlocks	1
RootInodeNumber	3
FreeInodes	21
FreeBlocks	20

Figura 39. Informació del *Superblock*

Aquesta captura ha estat feta després de crear només dos fitxers i escriure en ells 800 blocs de dades aproximadament.

Si seguim els mateixos passos que en la situació anterior tenim:

- 9 blocs inicials.
- 2 blocs dels inodes dels fitxers.
- 20 blocs que estan lliures.

Això fa un total de 1.610 blocs que són 6.594.560 bytes (1.610 blocs * 4.096 bytes/bloc).

Per altra banda, la mida total en bytes del *Softdisk* és 6.721.536 bytes (1.641 blocs * 4.096 bytes/bloc).

Un cop tenim que ocupa cada cosa trobem que l'eficiència en espai és la següent:

Eficiència en espai = dades usuari en bytes / mida *Softdisk* en bytes = 6.594.560 bytes / 6.721.536 bytes = 0.98 * 100 = 98 % d'eficiència en espai.

Cal tenir en compte que per aquest càlcul els bitmaps només ocupen 2 blocs entre els dos.

Per tant podem veure que, si tenim pocs fitxers però aquests contenen moltes dades, podem obtenir una eficiència en espai altíssima mentre que si tenim fitxers amb poques dades, l'eficiència estarà aproximadament en el 50 %.

9. Planificació i costos finals

A l'inici d'aquest projecte es va dur a terme una planificació que es pot observar a la figura 5.

La fase prèvia i inicial s'han seguit exactament com es va planificar en un primer moment sense cap tipus d'incidència.

En canvi, en la fase de desenvolupament s'ha produït una desviació (concretament en la Implementació de l'estructura interna del sistema de fitxers dinàmic) que es comenta a continuació.

A l'hora de fer l'anàlisi/especificació d'aquesta part es va realitzar amb les bases mal enteses per aquella part amb la qual cosa va endarrerir l'inici de les següents tasques una setmana.

Aquesta desviació no afecta els objectius d'aquest TFG que segueixen sent els que eren però els costos associats a aquest es veuen incrementats. Es veuen afectats els costos de dissenyador de la Fase de Desenvolupament - Implementació de l'estructura interna, la contingència dels costos directes i conseqüentment el cost final del projecte.

Si quantifiquem en hores i dies aquesta desviació obtenim 7 dies i 28 hores.

A continuació es mostren els nous costos:

Fase de Desenvolupament Implementació de l'estructura interna	Recursos	Hores	Cost
Especificació (Inicial)	Dissenyador	20	343,6 €
Especificació (Final)	Dissenyador	48	824,64 €

Taula 10. Costos Especificació

Per tant els costos directes es veuen incrementats en 481,04 €.

Costos	Percentatge	Preu	Cost
Directes	15 %	12.161.28 €	1.824,192 €
Indirectes	15 %	778,36 €	116,754 €
Total			1.940,946 €

Taula 11. Costos Totals Contingència

La contingència dels costos directes s'incrementa en 72.156 €.

El cost total és el següent:

Concepte	Cost
Costos directes	12.161,28 €
Costos Indirectes	778,36 €
Contingència	1.940,946 €
Imprevistos	841,48 €
Total	15.722,066 €

Taula 12. Costos Totals

A continuació es mostra la planificació final.

Nombre	Fecha de inicio	Fecha de fin
Fase Prèvia	5/09/16	7/09/16
Fase Inicial	9/09/16	6/11/16
Documentació GEP	9/09/16	24/10/16
Presentació GEP	31/10/16	6/11/16
Fase d'anàlisi	12/09/16	25/09/16
Fase de desenvolupament	3/10/16	1/01/17
Implementació del sistema...	3/10/16	30/10/16
Especificació	3/10/16	8/10/16
Implementació	9/10/16	24/10/16
Proves	26/10/16	27/10/16
Memòria	28/10/16	30/10/16
Implementació de l'estructu...	31/10/16	1/01/17
Especificació	31/10/16	13/11/16
Implementació	14/11/16	23/12/16
Proves	24/12/16	26/12/16
Memòria	27/12/16	1/01/17
Fase Final	2/01/17	23/01/17
Proves finals	2/01/17	8/01/17
Memoria	9/01/17	16/01/17
Preparació defensa oral	17/01/17	23/01/17

Figura 40. Planificació Final

10. Identificació de lleis i regulacions

En aquest treball no es considera el marc legal i això és perquè, en ser una part d'un projecte de la UPC, els aspectes legals s'han de tenir en compte en l'ús que se li donarà a aquest software un cop estigui dins del projecte de la UPC. En cap cas aquest treball incompleix cap normativa en utilitzar el software implementat.

11. Sostenibilitat i compromís social

A continuació es presenta l'estudi de sostenibilitat del projecte. La taula 8 mostrà les puntuacions obtingudes per cada apartat.

Sostenibilitat	Econòmica	Social	Ambiental
Valoració	8,3	7,7	8.2

Taula 13. Matriu de sostenibilitat

11.1 Sostenibilitat econòmica

S'ha realitzat una avaluació de costos tant de recursos materials com humans. Aquests costos s'han calculat tenint en compte que una sola persona és l'encarregada de dur a terme totes les tasques involucrades. Es podria realitzar el projecte amb molt menys temps amb un equip més gran o realitzant aquest projecte amb una dedicació de 8 hores diàries cosa que, actualment, és impossible a causa dels compromisos de feina. En el primer cas els costos incrementarien mentre que, en el segon, es veurien disminuïts. El temps dedicat a cada tasca és proporcional a la seva importància. Aquest projecte dóna suport a un projecte de la UPC per tant, millora les seves necessitats. Si el grup de recerca de la UPC decidís comercialitzar el producte, el podrien comercialitzar a "cost de fàbrica", ja que el producte a vendre serà 100% beneficis, per tant, el projecte és viable.

La sostenibilitat econòmica d'aquest projecte obté una nota de 8,3.

11.2 Sostenibilitat social

Actualment la política del nostre país està completament inestable perquè portem gairebé un any sense govern i anem cap a unes terceres eleccions. Aquest projecte està en el sector de software i és necessari perquè gràcies a ell es millorarà un projecte de la UPC. Naturalment aquest treball no ajudarà a la situació social/política del nostre país. Aquest projecte no canviarà la vida de l'usuari directament però si indirectament amb el projecte de la UPC. Cap col·lectiu es veurà perjudicat amb la realització d'aquest treball.

La sostenibilitat social d'aquest projecte obté una nota de 7,7.

11.3 Sostenibilitat ambiental

Pel que fa a sostenibilitat ambiental, només es veu afectada pels recursos indirectes que s'utilitzen en aquest projecte perquè el que es desenvolupa és software. El fet d'utilitzar RENFE pel desplaçament, consum d'electricitat pels ordinadors i l'adquisició d'aquests... és el que afecta la sostenibilitat ambiental. Quant al reciclatge no hi ha problema, simplement caldrà esborrar tot el codi relacionat amb aquest software.

La sostenibilitat social d'aquest projecte obté una nota de 8.2.

12. Conclusions

L'objectiu d'aquest projecte era crear un sistema de fitxers dinàmic que permetés incrementar-lo tant com volguéssim sense limitar-lo a una mida predeterminada.

Després de tota la informació donada podem afirmar que el projecte compleix aquest objectiu perquè s'ha provat que totes les funcions d'un sistema de fitxers funcionen amb la nova estructura interna del sistema de fitxers dinàmic. Hem comprovat després del test, que el sistema de fitxers pot créixer tant creant molts arxius com escrivint moltes dades. Podem assegurar també que la gestió dels bitmaps és correcte (que és robust el sistema implementat) i, a més a més, hem explicat l'eficiència en espai aconseguida.

Les grans dificultats d'aquest projecte han estat principalment entendre el codi del qual havia de partir que era molt extens (sempre és difícil programar a partir d'un codi no escrit per tu) i la implementació de la gestió dels bitmaps que ha donat bastants problemes. Tot i així, el primer obstacle s'ha pogut superar amb l'ajuda dels meus tutors que sempre han mostrat una predisposició per ajudar-me a entendre aquelles parts més complicades mentre que el segon, amb esforç i dedicació he aconseguit tirar endavant i deixar enrere els problemes que donava.

Tot i aquestes dificultats, s'ha aconseguit un sistema de fitxers dinàmic funcional i que podria ser utilitzat en un futur en el projecte de la UPC.

12.1 Treball futur

Abans d'iniciar aquest projecte, els directors em van comentar que el que volien era créixer i decréixer dinàmicament però a causa de la curta duració d'un quadrimestre es va decidir que l'objectiu del projecte seria que el sistema de fitxers creixés dinàmicament.

Per tant com a treball de futur, es pot implementar l'opció de decréixer dinàmicament quan esborréssim dades del sistema de fitxers perquè ocupi el mínim espai possible sense tenir blocs pel mig inutilitzats.

12.2 Conclusions personals

Acabat el projecte cal fer una valoració personal. Al començament del projecte tenia inseguretat en si aquest aniria bé o no degut a que el codi era completament desconegut i que no havia vist mai la implementació d'un sistema de fitxers, però la confiança que els directors, Alex Pajuelo i Xavier Verdú, van mostrar-me des de l'inici i el meu interès en els sistemes de fitxers, m'han ajudat a tirar endavant el projecte.

La valoració d'aquest treball és molt bona i ha valgut la pena. He pogut veure la importància de documentar el codi perquè ajuda molt més a entendre'l i no trobar-te amb línies sense sentit per tu. El fet de treballar amb repositoris i un codi no escrit per mi m'ha donat una experiència molt necessària per al món laboral perquè això es troba en totes les empreses. També m'ha ensenyat el dur que pot arribar a ser estar treballant en un problema i, en no ser capaç de solucionar-ho, ser capaç de canviar el punt de vista, prendre't les coses amb calma i veure quines són les possibilitats que poden estar causant els problemes. A més a més, m'ha ajudat molt a millorar la meva depuració que és tan important (o més) com programar.

Òbviament, també m'ha servit per comprovar que les coses no surten com un s'espera al principi i al final has de canviar les idees de com volies fer les coses.

Per acabar, només puc dir que estic satisfet amb la feina realitzada.

13. Bibliografía

1. Kubernetes. [En línia] [Data: 26 / Setembre / 2016.] http://kubernetes.io/docs/whatisk8s/.

2. Docker. [En línia] [Data: 26 / Setembre / 2016.] https://www.docker.com/what-docker.

3. The Linux Documentation Project. [En línia] [Data: 26 / Setembre / 2016.] http://www.tldp.org/pub/Linux/docs/ldp-archived/system-admin-guide/translations/es/html/ch06s08.html.

4. SlideShare. [En línia] [Data: 28 / Desembre / 2016.] http://www.slideshare.net/littleidea/container-crash-course/39.

5. Wikipedia. [En línia] [Data: 25 / Desembre / 2016.] https://en.wikipedia.org/wiki/LXC.

6. Nongnu. [En línia] [Data: 26 / Setembre / 2016.] http://www.nongnu.org/ext2-doc/ext2.html.

7. Wikipedia. [En línia] [Data: 26 / Setembre / 2016.] https://en.wikipedia.org/wiki/NTFS.

8. Microsoft. [En línia] [Data: 26 / Setembre / 2016.] https://msdn.microsoft.com/es-es/library/windows/desktop/aa365230(v=vs.85).aspx.

9. Suse. [En línia] [Data: 27 / Desembre / 2016.] https://www.suse.com/documentation/sles11/singlehtml/lxc_quickstart/lxc_quickstart.html#sec.lxc.startup.

10. Wikipedia. [En línia] [Data: 28 / Desembre / 2016.] https://en.wikipedia.org/wiki/Chroot.

11. man7.org/linux. [En línia] [Data: 27 / Desembre / 2016.] http://man7.org/linux/man-pages/man2/pivot_root.2.html.

12. GitLab. [En línia] [Data: 16 / Setembre / 2016.] https://about.gitlab.com/.

13. Gantt Project. [En línia] [Data: 1 / Octubre / 2016.] http://www.ganttproject.biz/.

14. Page Personnel. [En línia] [Data: 8 / Octubre / 2016.] http://www.pagepersonnel.es/sites/pagepersonnel.es/files/er_tecnologia16.pdf.

A. ANNEX

En aquest annex es pot veure el codi de la majoria de les funcions de les quals es parla en el capítol d'implementació de la memòria o d'algunes coses més específiques com la creació d'inodes especials o escriptures de bitmaps.

A.1 Creació Inode especial

```
BitmapBlocks = (struct Inode*)malloc((size_t)SBlock->BlockSize);
if (BitmapBlocks == NULL)
{
    free(SBlock);
    free(InodeBitmap);
    free(BlockBitmap);
    return WUSE_NOTENOUGHRESOURCES;
}

CreateInode(BitmapBlocks, 0, 1);
AddRefInode(BitmapBlocks);
BitmapBlocks->direct[0] = 5;
setBitmap(InodeBitmap, 0, 1);
```

A.2 Escriptura a disc d'inodes especials i dades

```
Werror = WriteBlock(sd, 2, (char*)BitmapBlocks);
if (Werror != WUSE_OK)
{
    free(InodeBitmap);
    free(BlockBitmap);
    free(SBlock);
    free(BitmapBlocks);
    free(BitmapInodes);
    free(LlistaInodes);
    SBlock = NULL;
    InodeBitmap = NULL;
    BlockBitmap = NULL;
    BitmapBlocks = NULL;
    BitmapInodes = NULL;
    LlistaInodes = NULL;
    return Werror;
}
```

A.3 Lectura Inode especial

```
BitmapBlocks = (struct Inode*)malloc((size_t)SBlock->BlockSize);

if (BitmapBlocks == NULL)
{
    free(SBlock);
    SBlock = NULL;
    return WUSE_NOTENOUGHMEMORY;
}

Werror = sd->Read(2 * SBlock->BlockSize / sd->getSectorSize(), SBlock->BlockSize / sd->getSectorSize(), (char*)BitmapBlocks);
```

A.4 Escriptura d'un bitmap

```
if (InodeBitmap != NULL)
{
    UINT64 BytesToWrite = SBlock->InodesBitmapSize;
    Werror = WriteFile(BitmapInodes, InodeBitmap, 0, &BytesToWrite);
    if (Werror != WUSE_OK) return Werror;
    free(InodeBitmap);
    InodeBitmap = NULL;
}
```

A.5 Lectura Inodes

```
UINT64 ext2dynamic::ReadInode(UINT64 inode_number, struct Inode **inode)
{
    struct Inode *buscat = (struct Inode*)malloc((size_t)SBlock->BlockSize);
    UINT64 BytesToRead = SBlock->BlockSize;
    UINT64 Werror = 0;
    if (inode_number < 3) {
        return bc->ReadBlock(inode_number + 2, (char**)inode);
    }
    else {
        Werror = ReadFile(LlistaInodes, (char*)buscat, SBlock->BlockSize * (inode_number - 3), &BytesToRead);
        if (Werror != WUSE_OK) {
            return Werror;
        }
        *inode = buscat;
    }
    return WUSE_OK;

}
```

A.6 Escriptura Inodes

```
UINT64 ext2dynamic::WriteInode(UINT64 inode_number, struct Inode *inode)
{
    UINT64 BytesToWrite = SBlock->BlockSize;
    UINT64 Werror = 0;
    if (inode_number < 3) {
        return bc->WriteBlock(inode_number + 2, (char*)inode);
    }
    else {
        Werror = WriteFile(LlistaInodes, (char*)inode, SBlock->BlockSize * (inode_number - 3), &BytesToWrite);
        if (Werror != WUSE_OK) return Werror;
    }
    return WUSE_OK;
}
```

A.7 Buscar Inodes lliures

```
UINT64 ext2dynamic::getFirstFreeInode(UINT64 *InodeNumber)
{
    UINT64 BytesToRead = SBlock->InodesBitmapSize;
    UINT64 BytesToWrite = SBlock->InodesBitmapSize;
    UINT64 Bytes = 3;
    UINT64 Werror = 0;
    UINT64 NumberOfInodes = Bytes * 8;

    if (InodeBitmap == NULL) {
        BytesToRead = SBlock->InodesBitmapSize;
        InodeBitmap = (char*)malloc((size_t)(SBlock->InodesBitmapSize));
        Werror = ReadFile(BitmapInodes, (char*)InodeBitmap, 0, &BytesToRead);
        if (Werror != WUSE_OK) return Werror;
    }

    if (SBlock->FreeInodes == 0) {
        BytesToWrite = SBlock->InodesBitmapSize;
        Werror = WriteFile(BitmapInodes, InodeBitmap, 0, &BytesToWrite);
        free(InodeBitmap);
        InodeBitmap = (char*)malloc((size_t)(Bytes));
        ZeroMemory(InodeBitmap, (size_t)(Bytes));
        Werror = WriteFile(BitmapInodes, InodeBitmap, SBlock->InodesBitmapSize, &Bytes);
        if (Werror != WUSE_OK) return Werror;
        SBlock->NumberOfInodes += NumberOfInodes;
        SBlock->FreeInodes += NumberOfInodes;
        SBlock->InodesBitmapSize = (SBlock->NumberOfInodes / 8) + 1;
        SBlock->InodesBMBlocks = 1 + (SBlock->InodesBitmapSize / SBlock->BlockSize);
        free(InodeBitmap);
        BytesToRead = SBlock->InodesBitmapSize;
        InodeBitmap = (char*)malloc((size_t)(SBlock->InodesBitmapSize));
        Werror = ReadFile(BitmapInodes, (char*)InodeBitmap, 0, &BytesToRead);
        if (Werror != WUSE_OK) return Werror;
    }

    /* Get a free Inode */
    if (getFirstFree(InodeBitmap, SBlock->NumberOfInodes, InodeNumber) == 0) return WUSE_NOFREEINODE;
    setBitmap(InodeBitmap, *InodeNumber, 1);
    SBlock->FreeInodes--;

    return WUSE_OK;
}
```

A.8 Allibera Inode

```
UINT64 ext2dynamic::setFreeInode(UINT64 InodeNumber)
{
    UINT64 BytesToRead = SBlock->InodesBitmapSize;
    UINT64 Werror = 0;

    if (InodeBitmap == NULL) {
        BytesToRead = SBlock->InodesBitmapSize;
        InodeBitmap = (char*)malloc((size_t)(SBlock->InodesBitmapSize));
        Werror = ReadFile(BitmapInodes, (char*)InodeBitmap, 0, &BytesToRead);
        if (Werror != WUSE_OK) return Werror;
    }
    SBlock->FreeInodes++;
    return setBitmap(InodeBitmap, InodeNumber, 0);
}
```

A.9 Buscar blocs lliures

```
UINT64 ext2dynamic::getFirstFreeBlock(UINT64 *BlockNumber)
{
    UINT64 Werror = 0;
    UINT64 Bytes = 3;
    UINT64 BytesToRead = SBlock->BlocksBitmapSize;
    UINT64 BytesToWrite = SBlock->BlocksBitmapSize;
    UINT64 NumberOfBlocks = Bytes * 8;

    if (BlockBitmap == NULL) {
        BytesToRead = SBlock->BlocksBitmapSize;
        BlockBitmap = (char*)malloc((size_t)(SBlock->BlocksBitmapSize));
        Werror = ReadFile(BitmapBlocks, (char*)BlockBitmap, 0, &BytesToRead);
        if (Werror != WUSE_OK) return Werror;
    }

    if (SBlock->FreeBlocks == 0) {
        BytesToWrite = SBlock->BlocksBitmapSize;
        Werror = WriteFile(BitmapBlocks, BlockBitmap, 0, &BytesToWrite);
        if (Werror != WUSE_OK) return WUSE_NOTENOUGHRESOURCES;
        Werror = sd->getMoreBlocks(SBlock->BlockSize, NumberOfBlocks);
        if (Werror != WUSE_OK) return WUSE_NOTENOUGHRESOURCES;
        SBlock->NumberOfSectors += (NumberOfBlocks * SBlock->BlockSize / SBlock->SectorSize);
        free(BlockBitmap);
        BlockBitmap = (char*)malloc((size_t)(Bytes));
        ZeroMemory(BlockBitmap, (size_t)(Bytes));
        Werror = WriteFile(BitmapBlocks, BlockBitmap, SBlock->BlocksBitmapSize, &Bytes);
        if (Werror != WUSE_OK) return WUSE_NOTENOUGHRESOURCES;
        SBlock->FreeBlocks += NumberOfBlocks;
        SBlock->NumberOfBlocks += NumberOfBlocks;
        SBlock->BlocksBitmapSize = (SBlock->NumberOfBlocks / 8) + 1;
        SBlock->BlocksBMBlocks = 1 + (SBlock->BlocksBitmapSize / SBlock->BlockSize);
        free(BlockBitmap);
        BytesToRead = SBlock->BlocksBitmapSize;
        BlockBitmap = (char*)malloc((size_t)(SBlock->BlocksBitmapSize));
        Werror = ReadFile(BitmapBlocks, (char*)BlockBitmap, 0, &BytesToRead);
        if (Werror != WUSE_OK) return Werror;

    }

    /* Get a free block */
    if (getFirstFreeFrom(BlockBitmap, SBlock->FirstDataBlock, SBlock->NumberOfBlocks, BlockNumber) == 0) return WUSE_NOFREEBLOCK;
    setBitmap(BlockBitmap, *BlockNumber, 1);
    SBlock->FreeBlocks--;

    return WUSE_OK;
}
```

A.10 Allibera bloc

```
UINT64 ext2dynamic::setFreeBlock(UINT64 BlockNumber)
{
    UINT64 Werror = 0;
    UINT64 BytesToRead = SBlock->BlocksBitmapSize;

    if (BlockBitmap == NULL) {
        BytesToRead = SBlock->BlocksBitmapSize;
        BlockBitmap = (char*)malloc((size_t)(SBlock->BlocksBitmapSize));
        Werror = ReadFile(BitmapBlocks, (char*)BlockBitmap, 0, &BytesToRead);
        if (Werror != WUSE_OK) return Werror;
    }
    SBlock->FreeBlocks++;
    return setBitmap(BlockBitmap, BlockNumber, 0);
}
```

A.11 Ampliació de blocs

```
int SoftDisk::getMoreBlocks(UINT64 BlockSize, UINT64 Blocks) {
    UINT64 Werror = 0;
    LARGE_INTEGER distance;


    distance.QuadPart = ((Blocks*BlockSize/this->SectorSize)*this->SectorSize);
    Werror = SetFilePointer(this->DiskFile, distance.LowPart, &distance.HighPart, FILE_END);
    if (Werror == INVALID_SET_FILE_POINTER)
    {
        CloseHandle(this->DiskFile);
        return WUSE_WINDOWSERROR;
    }

    Werror = SetEndOfFile(this->DiskFile);
    if (Werror == 0)
    {
        CloseHandle(this->DiskFile);
        return WUSE_WINDOWSERROR;
    }

    this->NumberOfSectors += (Blocks * BlockSize / this->SectorSize);
    union union_MasterBootRecord MBR;
    MBR.MBR.SectorSize = SectorSize;
    MBR.MBR.NumberOfSectors = this->NumberOfSectors;
    SetFilePointer(this->DiskFile, 0, NULL, FILE_BEGIN);
    this->Write(0, 1, (char*)&MBR);

    return WUSE_OK;
}
```

A.12 Format

```
UINT64 VirtualFileSystem::Format(SoftDisk *sd, UINT64 fs_type, UINT64 BlockSize)
{
    UINT64 Werror=0;

    switch (fs_type)
    {
    case FS_EXT2:
        myfs=(LogicalFileSystem*) LogicalFileSystem::CreateFileSystem(FS_EXT2);
        if (myfs==NULL)
            return WUSE_UNKNOWNFILESYSTEM;
        Werror=myfs->format(sd, BlockSize);
        delete myfs;
        return Werror;
        break;
    case FS_DYNAMIC_EXT2:
        myfs = (LogicalFileSystem*)LogicalFileSystem::CreateFileSystem(FS_DYNAMIC_EXT2);
        if (myfs == NULL)
            return WUSE_UNKNOWNFILESYSTEM;
        Werror = myfs->format(sd, BlockSize);
        delete myfs;
        return Werror;
        break;
    default:
        return WUSE_UNKNOWNFILESYSTEM;
    }
    return WUSE_OK;
}
```

A.13 Mount

```
UINT64 VirtualFileSystem::Mount(SoftDisk *sd, UINT64 fs_type)
{
    UINT64 myfs_type=fs_type;
    UINT64 Werror=0;

    myfs=NULL;

    if (fs_type==FS_AUTO)
    {
        Werror=Detect(sd, &myfs_type);
        if (Werror!=WUSE_OK)
            return Werror;
    }

    switch(myfs_type)
    {
    case FS_EXT2:
        myfs=(LogicalFileSystem*) new ext2();
        Werror=myfs->Mount(sd);
        if (Werror!=WUSE_OK)
        {
            delete myfs;
            return Werror;
        }
        break;
    case FS_DYNAMIC_EXT2:
        myfs = (LogicalFileSystem*) new ext2dynamic();
        Werror = myfs->Mount(sd);
        if (Werror != WUSE_OK)
        {
            delete myfs;
            return Werror;
        }
        break;
    default:
        break;
    }
```

A.14 Detect

```
UINT64 VirtualFileSystem::Detect(SoftDisk *sd, UINT64 *fs_type)
{
    BOOL correct = FALSE;
    UINT64 Werror = 0;

    myfs = (LogicalFileSystem*)LogicalFileSystem::CreateFileSystem(FS_EXT2);
    if (myfs == NULL)
        return WUSE_NOTENOUGHRESOURCES;
    Werror = myfs->detect(sd, &correct);
    delete myfs;
    if (Werror == WUSE_OK)
    {
        *fs_type = FS_EXT2;
        return WUSE_OK;
    }

    Werror = 0;

    myfs = (LogicalFileSystem*)LogicalFileSystem::CreateFileSystem(FS_DYNAMIC_EXT2);
    if (myfs == NULL)
        return WUSE_NOTENOUGHRESOURCES;
    Werror = myfs->detect(sd, &correct);
    delete myfs;
    if (Werror == WUSE_OK)
    {
        *fs_type = FS_DYNAMIC_EXT2;
        return WUSE_OK;
    }
    else
    {
        return Werror;
    }
    return WUSE_UNKNOWNFILESYSTEM;
}
```